3D US^X-ASIC Specification

for

Oldelft US^X-Platform

US^X-ASIC V_2 195G091 01

Specification Version: 2.2

Date: 15 December 2021

Authors:

Nikola Radeljic-Jakic (nradeljic@oldelft.nl)

Boris Lippe (blippe@oldelft.nl)

Index

1. Features

Layout

- Single ASIC supports 64 x 16 (1024) elements on a 180 µm x 190 µm pitch
- Larger array possible by tiling multiple ASICs
- Redistribution by post-processing allows other pitches (e.g. 180 µm x 180 µm) and the closing of gaps between tiled ASICs

Receive

- 64 groups of 16 beamformed piezo-electric input channels
- Low-Noise Amplifier for every element (NF of 3 dB for the whole chain)
- 20 dB programmable Time-Gain Compensation (TGC)
- True time delay beamforming using integrated sample-and-hold with an update rate of 25 MHz
- Maximum of 12 x 40 ns (480 ns) of true time delay with 20 ns resolution (5 ns when non-dynamic beamforming is selected)
- Dynamic delay profile for improved nearfield resolution
- Low-impedance line driver for every group

Transmit

- Two-level pulser (0 V / 100 V) for every element with typ.14 mA drive strength
- Virtual tripolar pulsing
- Harmonic distortion (< 35 dB)
- Pulse cancellation (> 35 dB)
- Every pulser can be set individually in steps of 20 – 50 ns
- Glitch-free voltage limiter protection circuit

Overall

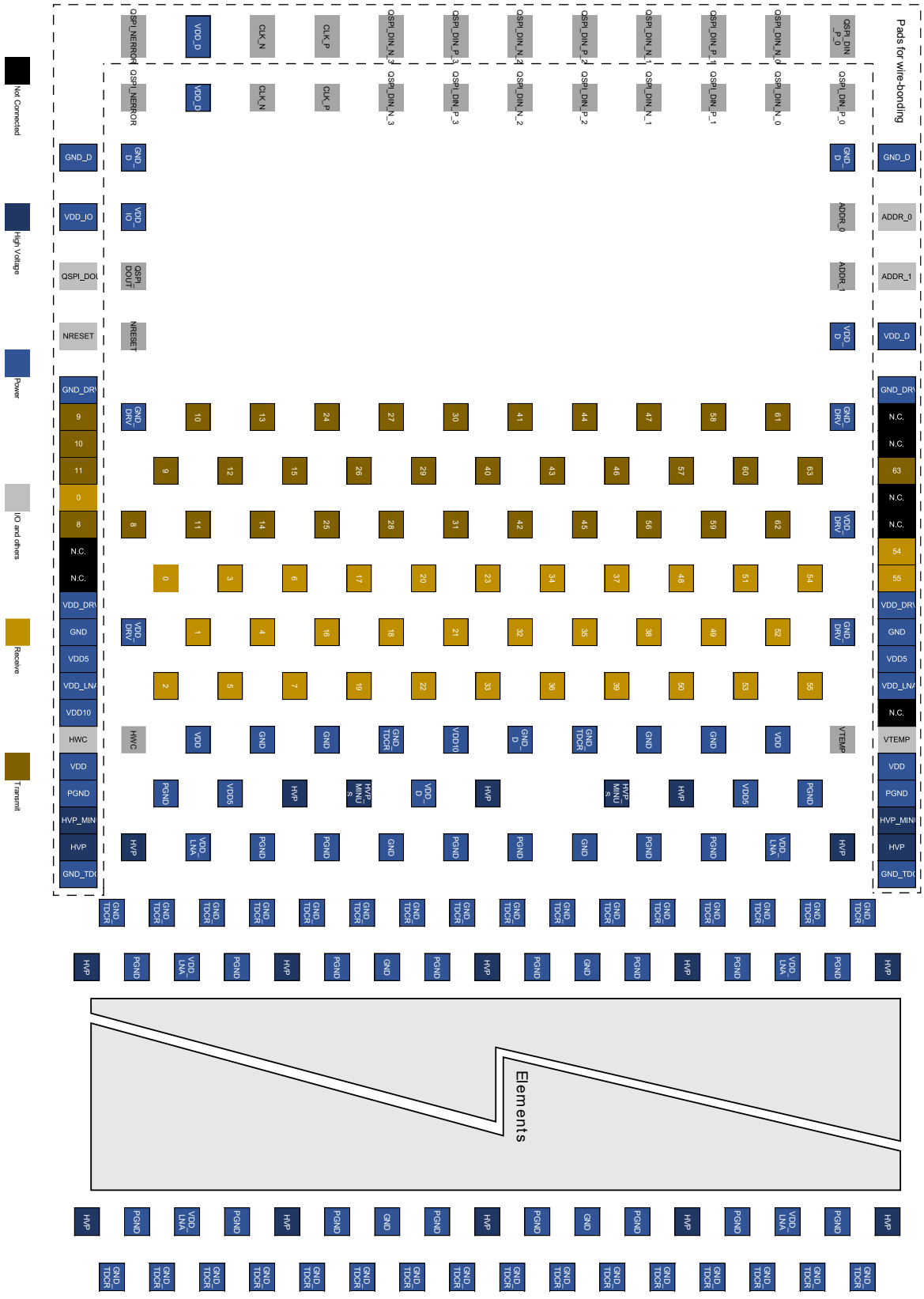
- Scalable power consumption of 220 – 1000 mW per ASIC
- 3.3 mm x 16 mm ASIC fabricated in XFAB XT018 0.18 µm HV -SOI Mixed Signal CMOS process

2. Description

The 3D US^{x}-ASIC is an integrated 1st stage analog front-end (AFE) for 2D-array ultrasound probes with integrated transmitters and receive beamforming. It is designed in such a way that multiple ASICs can be configured next to each other to form an array size between 3 mm x 11.5 mm and N x 3mm x 11.5 mm where N represents the number of ASICs. Configurable parameters allow the user to optimize between image quality, framerate and power consumption.

3. Pin Configurations

3.1. Pinout



3.2. Pin Descriptions

Table 1: Pin Descriptions.

Pad	Description	Type	Qty
V_{DD}	Analog Supply Voltage	A-PWR	2
$V_{DD,LNA}$	LNA Supply Voltage	A-PWR	4
$V_{DD,D}$	Digital Supply voltage	D-PWR	2
$V_{DD,IO}$	IO Supply Voltage	D-PWR	1
$V_{DD,DRV}$	Line-Driver Supply Voltage	A-PWR	4
V_{DD5}	Driver Voltage	A-PWR	2
V_{DD10}	Driver Voltage	A-PWR	1
HVP	Positive High Voltage	HV-PWR	4
HVP_{MINUS}	Driver High Voltage	HV-PWR	2
GND	Analog Ground	A-PWR	4
GND_D	Digital Ground	D-PWR	1
GND_{REF}	Temperature Sensor Reference Ground	A-PWR	1
GND_{DRV}	Line-Driver Ground	A-PWR	4
GND_{TDCR}	Transducer Ground	A-PWR	2
GND_P	Power Ground	HV-PWR	18
NRESET	Digital Reset (Active-Low)	CMOS IN	1
CLK_{ASIC}	Digital clock input (nom. 100 MHz)	LVDS IN	2
QSPI_DIN<3:0>	Digital data input	LVDS IN	8
QSPI_DOUT	Digital Data Output	CMOS OUT	1
QSPI_NERROR	Error Indicator (Active-Low)	CMOS INOUT[1]	1
ADDR<1:0>	ASIC Address	CMOS IN	2
SIG_CABLE<63:0>	µBeamformed Output Signals	A-OUT	64
V_{TEMP}	Temperature Sensor Output	A-OUT	1

[1] External pull-up resistor needed. Open-drain INOUT.

4. Electrical Ratings

4.1. Absolute Maximum Ratings

The values in Table 2 denote the absolute maximum ratings. These ratings are stress ratings only. Functional operation of the device is not guaranteed for these values. Please refer to section 4.2 for the recommended operating conditions.

Table 2: Absolute Maximum Ratings.

Symbol	Parameter	Min	Max	Unit
V_{DD}	Analog Supply Voltage	-0.5	2.3	V
$V_{DD,LNA}$	LNA Supply Voltage	-0.5	2.3	V
$V_{DD,D}$	Digital Supply voltage	-0.5	2.3	V
$V_{DD,IO}$	IO Supply Voltage	-0.5	7	V
$V_{DD,DRV}$	Line-Driver Supply Voltage	-0.5	2.3	V
V_{DD5}	Driver Voltage	-0.5	7	V
V_{DD10}	Driver Voltage for CW-Mode	-0.5	12	V
HVP	Positive High Voltage	-0.5	110	V
HVP_{MINUS}	Driver High Voltage	-0.5	110	V
GND_D GND_{REF} GND_{DRV} GND_{TDCR} GND_P	All Grounds	-0.5	0.5	V
$T_{STORE,MAX}$	Storage Temperature	-55	150	°C
$T_{J,MAX}$	Junction Temperature (with voltages applied)	-40	125	°C

4.2. Recommended Operating Conditions

Table 3: Recommended Operating Conditions.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{DD}	Analog Supply Voltage		1.71	1.8	1.89	V
$V_{DD,LNA}$	LNA Supply Voltage		1.71	1.8	1.89	V
$V_{DD,D}$	Digital Supply voltage		1.62	1.8	1.98	V
$V_{DD,IO}$	IO Supply Voltage		1.71	2.5	5	V
$V_{DD,DRV}$	Output-Driver Supply Voltage		1.71	1.8	1.89	V
V_{DD5}	Driver Voltage		4.75	5	5.25	V
V_{DD10}	Driver Voltage for CW-Mode		9	10	11	V
HVP*	Positive High Voltage		8	-	100	V
HVP	Positive High Voltage – current		-	-	10	mA
HVP	Positive High Voltage – power		-	-	500	mW
HVP_{MINUS}	Driver High Voltage	$HVP-HVP_{MINUS}$	4.75	5	5.25	V
GND_D GND_{REF} GND_{DRV} GND_{TDCR} GND_P	All Grounds		-0.2	0	0.2	V
$T_{STORE,MAX}$	Storage Temperature		0	40	85	°C
$T_{J,MAX}$	Junction Temperature (with voltages applied)		-40		125	°C
$P_{DIS,MAX}$	Power Dissipation		250	500	1000	mW

** For plane wave (all Tx delays equal), HVP must be <68 Volt for 50 ns pulse, <98 Volt for 100 ns pulse; otherwise digital errors may occur.*

4.3. ESD Ratings

All non-transducer ASIC pads have a 2 A discharge path to GND and are designed to meet the ESD-specifications listed in Table 4.

Table 4: The ESD-models that the ASIC was designed for.

MODEL		RATING
Human Body Model	(HBM)	> ± 2 kV
Machine Mode	(MM)	> ± 200 V
Charged Device Model	(CDM)	> 500 V

The transducer pads have only small protection structures and are very sensitive – they must not be touched by hand to prevent ESD damage. Latch-up is prevented by using a SOI technology.

4.4. Startup Conditions

It is strongly recommended to power up the low voltages (V_{DD}, $V_{DD,D}$, $V_{DD,IO}$, $V_{DD,LNA}$, $V_{DD,DRV}$) before powering up the higher voltages to ensure internal logic to be in a defined state. During power up of the ASIC the following relative potentials have to be maintained :

Table 5: Voltage Restrictions.

Restrictions
$V_{DD10} \geq V_{DD5}$
$HVP \geq HVP_{MINUS}$

NRESET and $CLK_{P,N}$ can be applied without taking into account any timing requirements.

4.5. Power Consumption

Table 6: Main power contributors in the US^X-ASIC with typical settings (B-Mode)[2].

Module	Calculation	Power Consumption [mW]	Amount	Total Power [mW]
LNA	30 µA * 1.8 V (setting 4)	0.0054	1024	55.30
Output Driver (Core)	50 µA * 1.8 V (setting 4)	0.0090	64	5.76
Output Driver (Peri)	219 µA * 1.8 V (setting 3)	0.394	64	25.23
Biasing circuitry	6 mA * 1.8 V	10.80	1	10.80
Digital TX RX SETUP STATIC	(Duty Cycle * I * V) 0.05 * 16 mA * 1.8 V 0.95 * 20 mA * 1.8 V 1.00 * 2 mA * 1.8 V 1.00 * 4 mA * 1.8 V	46.44	1	46.44
B-Mode Pulsing	2 pulses 40 V at 2.5 pF and 5 kHz PRF (N*C*V*V*f)	0.040	1024	40.96
Total[2]				**184**

Table 7: Power consumption of different pulsing modes[2].

Imaging Mode	Calculation $P=NCV^2f$	Power Consumption [µW]	Amount	Total Power [mW]
B-mode pulsing (unipolar, single)	1 pulse 40V at 2.5 pF and 2.5kHz PRF	10	1024	10.24
B-mode pulsing (unipolar, single)	1 pulse 40V at 2.5 pF and 5kHz PRF	20	1024	20.48
B-mode pulsing (unipolar, double)	2 pulse 40V at 2.5 pF and 5kHz PRF	40	1024	40.96
B-mode pulsing (tripolar)	3 Half-cycles 40V at 2.5 pF and 5kHz PRF	45	1024	46.08
B-mode pulsing (tripolar)	3 Half-cycles 30V at 2.5 pF and 5kHz PRF	27.5	1024	28.16
PW mode	10 pulse 30V at 2.5 pF and 5kHz PRF	112.5	1024	115.20

4.6. Capacitors at supply lines

Capacitors must be placed close to the ASIC; see Table 8. For HVP, 100 nF is the minimum capacitance. Increasing the capacitance to 250 nF gives a better performance in terms of pulsing power homogeneity.

[2] Values are an indication; a detailed power model is available.

Table 8: Minimum capacitance at supply lines (per ASIC).

Symbol	Name	Capacitance
V_{DD}	Analog Supply Voltage	1 µF
$V_{DD,LNA}$	LNA Supply Voltage	1 µF
$V_{DD,D}$	Digital Supply voltage	1 µF
$V_{DD,IO}$	IO Supply Voltage	100 nF
$V_{DD,DRV}$	Output-Driver Supply Voltage	1 µF
V_{DD5}	Driver Voltage	1 µF
V_{DD10}	Driver Voltage for CW-Mode	1 µF
HVP	Positive High Voltage	100 nF (recommended: 250 nF)
HVP_{MINUS}	Driver High Voltage	100 nF

4.7. Decoupling capacitors in signal cables

Decoupling capacitors are required in the signal cables SIG_CABLE<63:0> before connecting to the system. The recommended value is 10 nF for each cable.

5. Block Diagram

5.1. Schematic Block Diagram

A schematic block diagram of the ASIC is shown in Figure 1. The ASIC is divided into two main parts:

- Core
- Periphery

The core contains:

- 64 groups, consisting of:
  - 16 ‘Single Channels’ each containing:
    - Transducer pad to connect a transducer element
    - Two-level pulser for pulsing up to 100 V
    - Voltage limiter circuits to protect the receive circuitry from the high voltages
    - Time-gain compensation to attenuate strong near field signals
    - Low-noise amplifier signal gain
  - 1 µBeamformer for on-chip receive beamforming
  - 1 output driver for driving the combined signal to the periphery
  - Setup registers
- Waveform generator

The periphery contains:

- 64 output drivers to drive all 64 acoustic lines simultaneously
- Bias circuitry and calibration options
- Testing circuitry
- Fuse memory to write serial numbers
- Setup registers
- Communication interface
- Digital blocks for beam timing, receive- and transmit-clock generation

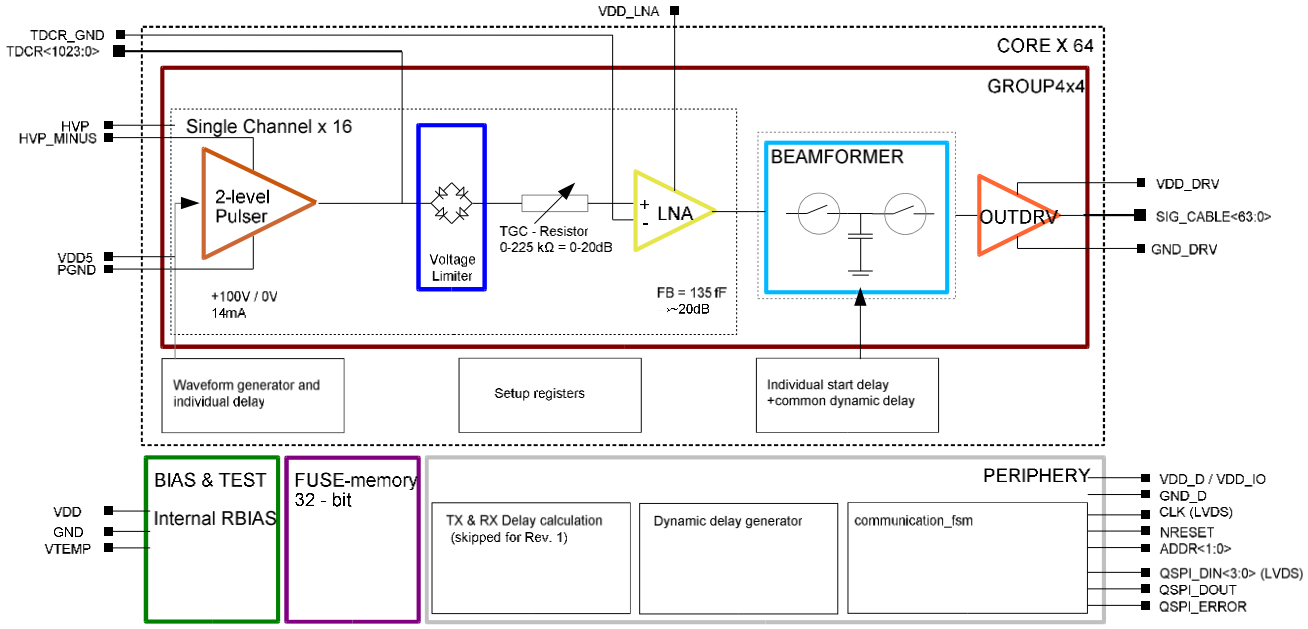


Figure 1: Schematic block diagram of the ASIC.

5.2. Layout Block Diagram

A floorplan of the ASIC is shown below in Figure 2 where the size of the ASIC, groups and elements can be seen.

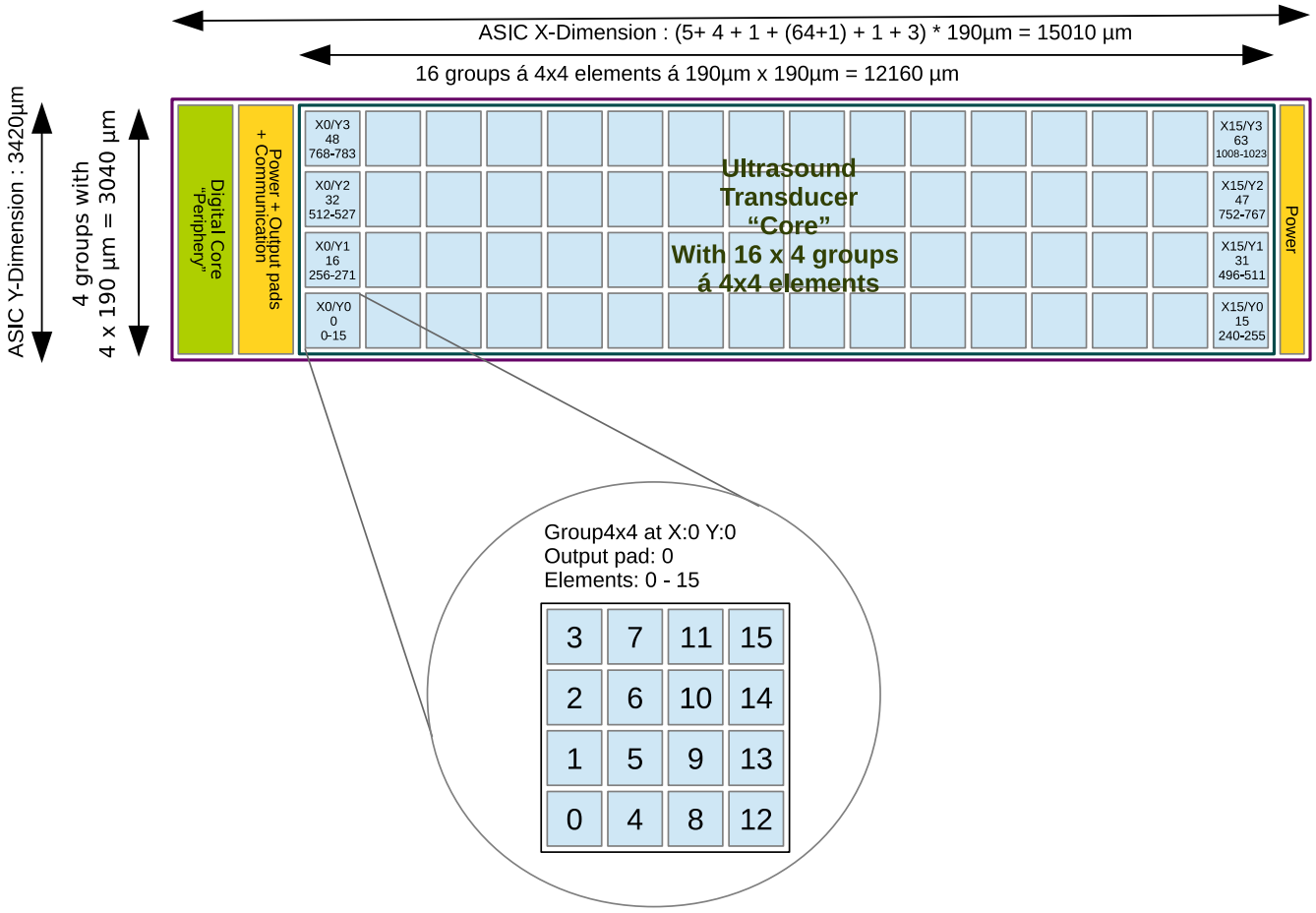


Figure 2: Floorplan of the ASIC.

6. Communication Interface

The ASIC contains a modified Serial Peripheral Interface (SPI) with variable transmission length for communication. The SPI contains a clock line, 4 parallel MOSI lines (QSPI_DIN<3:0>), 1 MISO line (QSPI_DOUT), error indicator and a 2-bit address. A block diagram of the interface is shown in Figure 3.

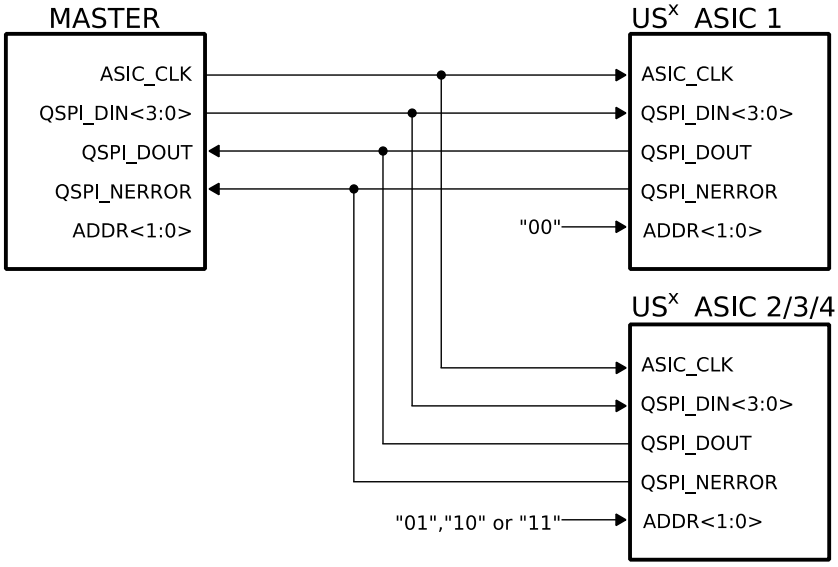


Figure 3: USX-ASIC communication interface with multiple ASICs.

More information on the shown interface is listed below in Table 9.

Table 9: USX-ASIC communication interface signals.

SIGNAL	I/O	TYPE	# PINS	NOTES
ASIC_CLK	IN	LVDS	2	Nom. 100 MHz.
QSPI_DIN<3:0>	IN	LVDS	8	Total transfer speed of 400 Mbit/s for a 100 MHz clock frequency.
QSPI_DOUT	OUT	CMOS	1	Nom. 100 MHz, can be lowered by altering the 'DataOutClkDiv' value in the BIST register. Can operate in both a low- and high-impedance state by configuring the BIST register.
QSPI_NERROR	OUT	CMOS	1	Active-low error indicator; open-drain output
ADDR<1:0>	IN	CMOS	2	ASIC address integrated on a PCB/FPC, used to address each ASIC individually via the 'AddrMask' command described below.

6.1. Device Addressing

Every ASIC has two pins (ADDR<1:0>) that define its binary encoded address. These pins are connected to either GND/VDD on the underlying PCB/FPC to produce one of four possible combinations: "00","01","10" or "11". This allows the user to address and thus write data to ASICs individually. The addressing is done by sending the 'AddrMask' command with the correct one-hot encoded bitmask. More information on the bitmask is shown in Table 10.

Table 10: AddrMask command for device addressing.

COMMAND	CMD Code	NOTES
AddrMask	0x06 - 0xF6	Command used to enable communication to the ASIC. The high nibble defines the one-hot encoded bitmask to address ASICs in an array configuration. (e.g. 0xF6 addresses all possible ASICs, 0x26 addresses only the second ASIC.)

6.2. Clocking

The US^X-ASIC requires a differential clock input to operate correctly. The SPI utilizes this same clock to communicate with the US^X-ASIC. The clock requirements can be found in section 8.3.3. The nominal clock frequency is 100 MHz and all the timing specifications in the remainder of this chapter is based on this frequency.

6.3. Write Operations

A write operation to the US^X-ASIC consists of three parts. First, the 'AddrMask' command with the correct bitmask has to be sent in order to enable communication with the correct ASIC. Second, the command is sent directly after the 'AddrMask' command. Finally, a 1-byte CRC-8 value is added to the write operation to verify that the data has arrived correctly. If the data has not arrived correctly, a QSPI_ERROR event will be triggered.

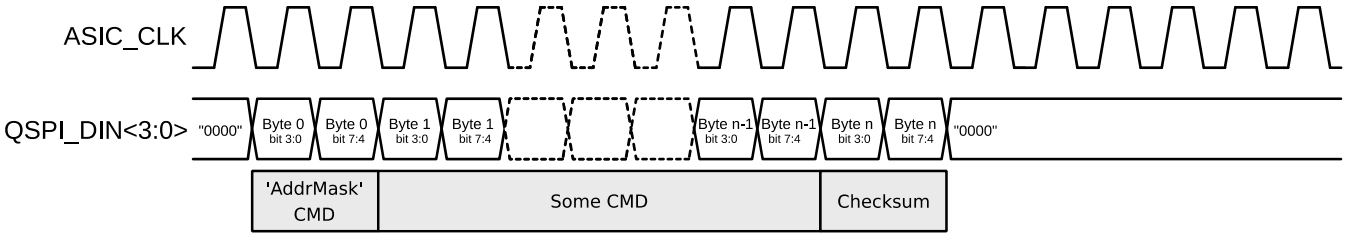


Figure 4: Timing diagram for writing to the ASIC.

6.4. Read Operations

A read operation from the US^X-ASIC first requires a 'Read' command to be written to the ASIC. Multiple 'Read' commands exist that all read out different registers from the ASIC. 'Read' commands are also preceded by the 'AddrMask' command. When a 'Read' command has arrived to the ASIC, the first byte that is returned is the same 'Read' command, followed by the data stored in the targeted register. The data is followed by a 1-byte CRC-8 value, calculated over the complete return data.

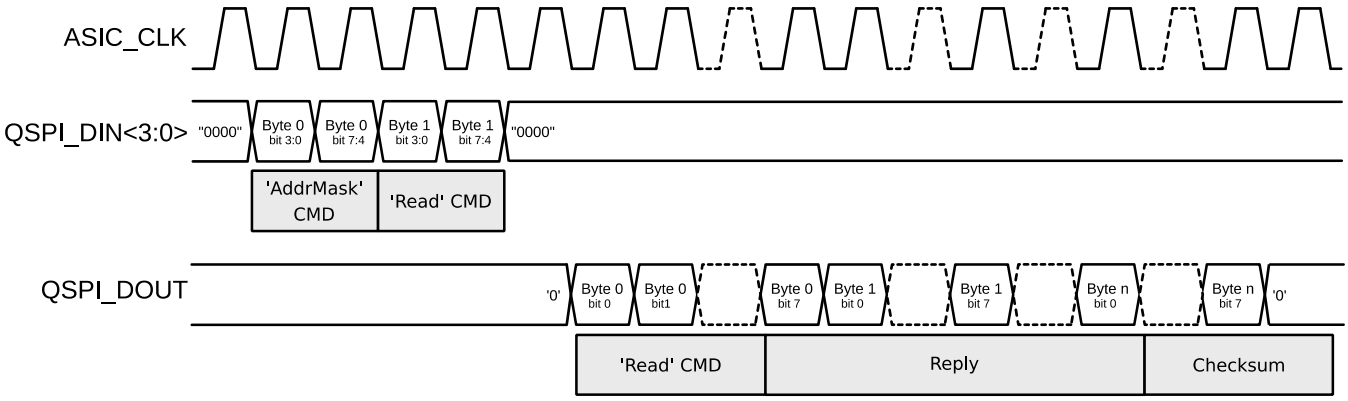


Figure 5: Timing diagram for reading from the ASIC.

6.5. Command Overview

The available commands can be categorized into three types that will be listed in the following sections:

- Start commands
- Short commands
- Long commands

6.5.1. Start Commands

To make the ASIC more resilient against spikes on the data lines, data needs to be made valid by sending the 'AddrMask' command. The NOP command can always be sent to the ASIC even if the 'AddrMask' command has not be sent before. All other commands require the 'AddrMask' command to be sent first, otherwise the error line will be asserted.

Table 11: Start Commands.

Command	CMD Code	Send Bytes	Time in µs	Description
NOP	0x00	1	0.02	Default bus state before the 'AddrMask' command, can also be inserted between the 'AddrMask' command and any other command to meet timing requirements.
AddrMask	0x06 – 0xF6	1	0.02	Command used to enable communication to the ASIC. The high nibble defines the bitmask to address ASICs in an array configuration. (e.g. 0xF6 addresses all possible ASICs, 0x26 addresses only the second ASIC.)

6.5.2. Short Commands

Short commands consist of the following two bytes and have no checksum or receive data:

- Timing is based on a 100 MHz write clock
- Every send-byte requires 2 clock cycles to write (20 ns)
- The following data is included in the Send Bytes count:
  - 'AddrMask' CMD (1 byte)
  - CMD code (1 byte)
  - NOP (1 byte)
- Overall time per command is calculated as follows:
  - (Send Bytes) * 20 ns

Figure 6: Short command structure.

Table 12: Short commands.

Command	CMD Code	Send Bytes	Time in µs	Description
ClearError	0xD2	3	0.06	Clear the external error pin QSPI_ERROR.
ClearCore	0xEB	3	0.06	Reset the configuration of all groups immediately. → fast power down
StartBistADC	0x9A 0x9D	3	0.06	Start a single conversion with the BIST-ADC. The BIST_CONFIG setup can be used the set up the ADC. Either command can be used, they have same functionality.
TriggerStop	0xF3	3	0.06	Reset the RX clock generator, stopping all clocks.
TriggerPage00, TriggerPage01, TriggerPage10, TriggerPage11	0x83 0x93 0xA3 0xB3	3	0.06	RX and TX clock generator; set up all element delays in all groups for receive and transmit with the settings stored in the following registers: RXDELAY0, RXDELAY1, RXDYN, TXDELAY0, TXDELAY1 TriggerPage00 → TXDELAY0 & RXDELAY0 TriggerPage01 → TXDELAY0 & RXDELAY1 TriggerPage10 → TXDELAY1 & RXDELAY0 TriggerPage11 → TXDELAY1 & RXDELAY1 If 'BeamWaitRun' in 'REG_CONFIG' is set, the 'TriggerRun' command must be sent to start a beam. Alternatively, the beam starts immediately after the setup is complete.
TriggerRun	0xE3	3	0.06	Start the beam, which is on 'hold'
FuseWriteStart	0x8A	3	0.06	Trigger writing one fuse byte. (See Table 42 for details.)

6.5.3. Long Commands

Just like short commands, long commands start with a 'AddrMask' command and CMD code. For long write commands however, the write data is added after the CMD code, followed by a CRC. For read operations, data returns from the ASIC starting with the sent CMD code, followed by the read data and CRC.
The naming convention is the following:

- Periphery registers are written/read using long commands starting with 'WriteRegister' and 'ReadRegister'. These refer to the registers with the prefix 'REG_' (Section 7.1).
- Core registers are written/read using long commands starting with 'WriteChain' and 'ReadChain'. These refer to the registers with the suffix '_DFF' (Section 7.2). A chain of 64 registers is written at once; they are scrambled before being sent. The output driver configuration 'CONFIG_DRV' is also written/read in a chain of 64 registers. CONFIG_DRV data is scrambled, but another scrambling algorithm is used than for the core registers _DFF.

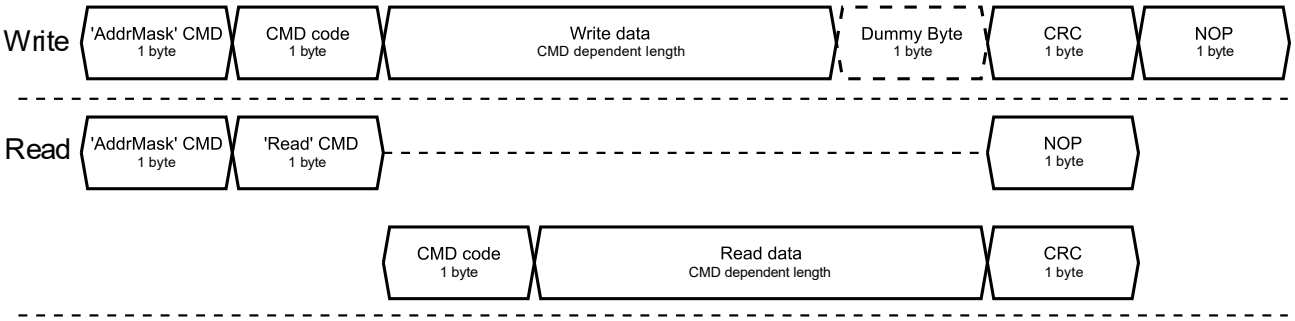


Figure 7: Long command structure for write and read operations.

The long commands are listed in Table 13. Some notes on the table and on Figure 7:

- Timing is based on a 100 MHz write clock, with a return data rate of 12.5 MHz ('DataOutClkDiv'= 7).
- Every send-byte requires 2 clock cycles to write (20 ns)
- Every return-byte requires 8 clock cycles to read (8 x 80 ns = 640 ns)
- The following data is included in the Send Bytes count:
  - 'AddrMask' CMD (1 byte)
  - CMD code (1 byte)
  - Write data (command dependent length)
  - Dummy byte (1 byte for core registers and CONFIG_DRV only)
  - CRC (1 byte for write commands only; calculated over AddrMask, CMD code, write data and dummy byte)
  - NOP (1 byte)
- The following data is included in the Return Bytes count for read commands only[3]:
  - CMD code (1 byte)
  - Read data (command dependent length)
  - CRC (1 byte) (calculated over AddrMask + CmdCode and read data)
- For Read, the QSPI_DIN lines must be set low (writing a NOP) after the Read CMD. The lines must be low when CRC is returned; between the Read CMD and the returning CRC the levels on the QSPI_DIN do not care.
- Overall time per command is calculated as follows:
  - (Send Bytes) * 20 ns + (Return Bytes) * 8 * 80 ns
- The transmission speed of the return data can be reduced by changing the 'DataOutClkDiv' value in the BIST register. If changed, the timing for the return data needs to be adapted accordingly.

[3] Write commands don't return any data.

Table 13: Long commands.

Command	CMD Code	Send Bytes	Return Bytes	Time in µs	Description
WriteRegisterCONFIG	0x04	11	0	0.22	Write periphery registers[2].
WriteRegisterCERXTXCONF	0x14	9	0	0.18	
WriteRegisterRXDELAY0	0x24	20	0	0.40	
WriteRegisterRXDELAY1	0x34	20	0	0.40	
WriteRegisterRXDYN	0x44	25	0	0.50	
WriteRegisterTXDELAY0	0x54	12	0	0.24	
WriteRegisterTXDELAY1	0x64	12	0	0.24	
WriteRegisterBIST	0x74	8	0	0.16	
WriteRegisterTIMING	0x84	13	0	0.26	
ReadRegisterCONFIG	0x05	2	9	5.82	Read periphery registers.
ReadRegisterCERXTXCONF	0x15	2	10	6.44	
ReadRegisterRXDELAY0	0x25	2	18	11.58	
ReadRegisterRXDELAY1	0x35	2	18	11.58	
ReadRegisterRXDYN	0x45	2	23	14.78	
ReadRegisterTXDELAY0	0x55	2	10	6.46	
ReadRegisterTXDELAY1	0x65	2	10	6.46	
ReadRegisterBIST	0x75	2	7	4.52	
ReadRegisterTIMING	0x85	2	11	7.10	
WriteChainSELECT	0x66	21	0	0.42	Write register chain in the core[1,2,3].
WriteChainMATRIX	0x56	3813	0	76.26	
WriteChainTXDELAY_DFF	0x46	709	0	14.18	
WriteChainRXDELAY_DFF	0x36	1029	0	20.58	
WriteChainCWSEL	0x26	517	0	10.34	
WriteChainXMITWF	0x16	517	0	10.34	
WriteChainCONFIG	0x06	325	0	6.50	
WriteChainCondMATRIX	0x5C	243	0	4.86	Write only the register chains, selected in the SELECT_DFF register.
WriteChainCondTXDELAY_DFF	0x4C	49	0	0.98	
WriteChainCondRXDELAY_DFF	0x3C	69	0	1.38	
WriteChainCondCWSEL	0x2C	37	0	0.74	
WriteChainCondXMITWF	0x1C	37	0	0.74	
WriteChainCondCONFIG	0x0C	25	0	0.50	
ReadChainSELECT	0x67	2	18	11.58	Read register chain from the core[3].
ReadChainMATRIX	0x57	2	3810	1219.90	
ReadChainTXDELAY_DFF	0x47	2	706	451.90	
ReadChainRXDELAY_DFF	0x37	2	1026	656.70	
ReadChainCLKCWSEL	0x27	2	514	329.02	
ReadChainXMITWF	0x17	2	514	329.02	
ReadChainCONFIG	0x07	2	322	206.14	

Command	CMD Code	Send Bytes	Return Bytes	Time in µs	Description
WriteChainDRV	0x08	133	0	2.66	Write output driver configuration[1,2,3]
ReadChainDRV	0x09	2	130	83.26	Read output driver configuration[3]
BistGetADCResult	0xAA	2	4	2.62	Read ADC conversion result
BistGetErrorStatus	0xBA	2	3	2.62	Read error status
BistGetFuseMemory	0xCA	2	6	3.90	Read fuse memory

[1]Dummy byte included.
[2]NOP included.
[3]Scrambling and descrambling needed; see Figure 9 and Figure 11.

6.6. CRC

All long commands sent to the ASIC are followed by an 8-bit (1 byte) CRC that is calculated as CRC-8 (Polynomial: x8 + x4 + x3 + x2 + 1 (also known as SAE-J1850)) with an initial CRC value of 0x2A. This CRC must be sent correctly to the ASIC in order to not trigger a QSPI_ERROR event.
All return data from the ASIC is also followed by the same 8-bit CRC, calculated over the complete return data. It can used to verify the integrity of the returned data.

The following C-code may be used to generate the correct CRC:

```
unsigned char asic_driver::calc_crc8(unsigned char *send_string, int size)
{
   static unsigned char crc8_table[256];    /* 8-bit table */
   static int made_table=0;
   int i,j;
   unsigned char crc;

   // CRC-8 (SAE-J1850) (Polynomial : x8 + x4 + x3 + x2 + 1) with Initial value 42 and swapped nibbles
   #define CRC_POLY  0x1D
   if (!made_table)
   {
      for (i=0; i<256; i++)
      {
         crc = i;
         for (j=0; j<8; j++)
            crc = (crc << 1) ^ ((crc & 0x80) ? CRC_POLY : 0);
         crc8_table[i] = crc & 0xFF;
      }
      made_table=1;
   }
   crc=42;
   for (i=0; i< size; i++)
   {
      unsigned char data = ((*send_string >> 4) & 0x0F) | ((*send_string << 4) & 0xF0);
      crc = crc8_table[crc ^ data];
      send_string++;
   }
   return crc;
}
```

6.7. Error Handling

The ASIC contains a basic error handling scheme that pulls down the QSPI_ERROR line whenever an error occurs. Furthermore, the ASIC provides additional information about the root of the error in the ASIC Error Status Register listed in Table 14. When an error occurs and the QSPI_ERROR line is pulled low, all communication to the ASIC is ignored for 40.96 µs (length of the longest command). The error bits can be read back with the 'BistGetErrorStatus' command. The QSPI_ERROR line can only be cleared by sending the 'ClearError' command or by resetting the ASIC.

Table 14: ASIC Error Status Register.

BIT	ERROR	DESCRIPTION
0	UNKNOWN_CMD	Unknown command
1	VALIDERROR	Start/End of command is not valid
2	CHKSUMERROR	Receive checksum error at the end of a command
3	BUSY	Command received, but cannot be executed because the internal resource is busy
4	VERSION_BIT0	Returns the ASIC version number when 'ErrorGetVersion' in the REG_BIST register is set to '1'. (Not an error indicator). (0 = v1, 1 =v2, etc.)
5	VERSION_BIT1	
6	LOCKED	The user can disable all commands (except for the 'BeamStop' command) during receive, by sending the 'AddrMask' command with bitmask '0'. In that case, any command received will trigger this error.
7	EXTERNAL	External error received (QSPI_ERROR) held low externally

All error sources can be masked individually. If not masked, the error sources serve two purposes:

- Notifying the user that an error has occurred by pulling the QSPI_ERROR line low
- Performing one of the following 4 actions:
  - Stopping the receive operation
  - Stopping the transmit operation
  - Clearing the core registers
  - Clearing the output drivers

When the QSPI_ERROR goes low, during a command that does not involve the writing or reading of (additional) data to the ASIC (e.g. transmit delays), the user may assume that the command is simply ignored. When however, the user sends a command to Read/Write data (e.g. transmit delays) into the ASIC, the user must assume that the content of this whole register chain is now undefined. The register chains are serial shift registers that are cycled during Read/Write commands and must be re-written if this process is interrupted by an error.

7. Memory Overview

7.1. Periphery

Periphery registers (all have the prefix “REG_”) are located in the periphery of the ASIC. They contain common configuration data and internal test settings and results. The set of periphery registers is listed in Table 15.

Table 15: Periphery Registers.

Register	Length in bits	Description
REG_BEAMTIMING	72	Timing configuration of the internal transmit and receive circuitry
REG_BIST	40	All non-ultrasound functionality of the ASIC is configured here, mainly for the Built-In Self-Test ADC that is used to check operating points and capacitances. Also contains some global ASIC configuration options.
REG_TXDELAY1	64	Two sets of transmit delay coefficients, allowing the user to transfer new coefficients to one set, while calculations are being performed on the other set.
REG_TXDELAY0	64	
REG_RXDYN	168	Clock configuration for dynamic receive.
REG_RXDELAY1	128	Two sets of receive delay coefficients, allowing the user to transfer new coefficients to one set, while calculations are being performed on the other set.
REG_RXDELAY0	128	
REG_CERXTXCONF	64	Receive/Transmit element enable for aperture definition
REG_CONFIG	56	Various configuration parameters

7.1.1. REG_BEAMTIMING

Table 16: REG_BEAMTIMING.

Address	# Bits	Name	Description	Unit	Range
<7:0>	8	AnaResetStopTime	Sets the duration of the reset phase if the parameter 'UseAnalogReset' is asserted.	80 ns	0 – 20.40 µs
<15:8>	8	SetupRxTime	Sets the time at which the receive delays are loaded from the memory into the array. The loading of the receive delays takes 1.20 µs.	40 ns	0 – 10.20 µs
<23:16>	8	RunTxTime	Sets the start of the transmit phase.	40 ns	0 – 10.20 µs
<31:24>	8	RunRxTime	Sets the start of the receive phase. Must be >=SetupRxTime+29.	40 ns	0 – 10.20 µs
<39:32>	8	StopTxTime	Sets the end of the transmit phase.	80 ns	0 – 20.40 µs
<47:40>	8	StopRxTime	Sets the end of the receive phase after which the ASIC returns to an idle state.	2.56 µs	0 – 652.8 µs
<51:48>	4	TGCIncrement	Time between TGC gain steps. 0 and 1 are invalid.	640 ns	1.28 – 9.60 µs
<55:52>	4	TGCStart	Sets the start of the TGC curve.	1.28 µs	0 – 19.20 µs
<59:56>	4	TGCDecrement	Time between opening TGC switches. 0 is invalid.	20 ns	20 ns – 300 ns
<62:60>	3	TGCDecrementCnt	Number of switches that are opened at the final gain step as a means to compensate the charge injected by the final switch.	N	0 - 7
<63>	1	ExtRxSetup	Increases the available time for µBeamformer initialization from 1.20 µs to 2.32 µs. Required to dither the µBeamformer artifacts.	-	-
<64>	1	ExtTxRunTime	When '1' the TxRunTime is referenced to 'TriggerPage, when '0' to the end of the calculations. **Note: when '1', REG_CONFIG. BeamWaitRun=1 does not work.**	-	-
<65>	1	ExtRxSetupTime	When '1' the RxSetupTime is referenced to 'TriggerPage, when '0' to the end of the calculations.	-	-
<71:66>	6	-	Reserved	-	-

7.1.2. REG_BIST

Table 17: REG_BIST.

Address	# Bits	Name	Description	Range
<7:0>	8	ClearTime	Time to clear integration capacitor of the single slope converter in 160 ns steps	40.96 µs max.
<15:8>	8	ConfigIn	Bits <3:0> negative input select: 10-- : select one of CW_CH_RX<3:0> 11-- : select one of CW_CH_TX<3:0> 0X01 : select V_{BG} 0X10 : select V_{TEMP} Bits <7:4> positive input select: 10XX : select one of CW_CH_RX<3:0> 11XX : select one of CW_CH_TX<3:0> 0--- : select Capacitor 16pF	
<16>	1	DOUTAutoDis	Put the QSPI_DOUT driver in a high-impedance state	
<17>	1	DOUTAlwaysEn	Put the QSPI_DOUT driver in a constant low-impedance state	
<21:18>	4	DataOutClkDiv	Clock divider for the return data. The clock frequency is calculated by: $f_{clk,return} = \frac{100\ MHz}{1+DataOutClkDiv}$	100 MHz – 6.25 MHz
<23:22>	2	iVal	Test-current on the positive input of the BIST module 00 : disabled 01 : 0.5 µA 10 : 2.5 µA 11 : 10 µA	
<24>	1	ErrorGetVersion	If set to '1': the 'BistGetErrorStatus' command returns the ASIC version on bits <5:4> of 'Error_NMask_Out' in the REG_CONFIG register. If set to '0': bits <5:4> of 'Error_NMask_Out' in the REG_CONFIG register are unused and are always "00".	
<28:25>	4	IbiasCal	ASIC reference current calibration 4-bit unsigned code with inverted top bit: 1000 : min. current (approx. -30%) 0000 : typ. current reference 10uA 0111 : max. current (approx. +30%)	-30% - +30% of uncalibrated value
<29>	1	BistEn	Enable the BIST comparator	
<30>	1	-	Reserved	
<31>	1	FuseWriteEn	Enable the writing of fuses	
<39:32>	8	-	Reserved	-

7.1.3. REG_TXDELAY1 and REG_TXDELAY0

In order to reduce the setup time of the transmit delays, an approximation of a third order polynomial function is implemented in the ASIC. The coefficients of this polynomial can be configured in REG_TXDELAY1 and REG_TXDELAY0, such that one set can be loaded while the other is being decompressed.

Table 18: REG_TXDELAY1 and REG_TXDELAY0.

Address	# Bits	Name	Description
<7:0>	8	c_0	Polynomial coefficients from which the transmit delays are generated in units of $0.5 \cdot TX_{CLK}$ period. c_0 is an unsigned character [0 to 255] and $c_{1..7}$ are signed characters [-128 to 127].
<15:8>	8	c_1	
<23:16>	8	c_2	
<31:24>	8	c_3	
<39:32>	8	c_4	
<47:40>	8	c_5	
<55:48>	8	c_6	
<63:56>	8	c_7	

7.1.4. REG_RXDELAY1 and REG_RXDELAY0

In order to reduce the setup time of the receive delays, an approximation of a second order polynomial function is implemented in the ASIC. The coefficients of this polynomial can be configured in REG_RXDELAY1 and REG_RXDELAY0, such that one set can be loaded while the other is being decompressed.

Table 19: REG_RXDELAY1 and REG_RXDELAY0.

Address	# Bits	Name	Description
<7:0>	8	c_0	Polynomial coefficients from which the receive delays are generated in units of $0.5 \cdot RX_{CLK}$ period. c_9 is an unsigned character [0 to 255] and $c_{0..8}$ are signed characters [-128 to 127].
<15:8>	8	c_1	
<23:16>	8	c_2	
<31:24>	8	c_3	
<39:32>	8	c_4	
<47:40>	8	c_5	
<55:48>	8	c_6	
<63:56>	8	c_7	
<71:64>	8	c_8	
<79:72>	8	c_9	
<127:80>	48	-	Reserved

7.1.5. REG_RXDYN

Table 20: REG_RXDYN.

Address	# Bits	Name	Description	Unit
<2:0> <3> <6:4> <7>	3 1 3 1	Multiplier<0> MultiSign<0> StartPhase<0> Reserved	Multiplication factor of the master curve When '1' the delay increases, when '0' the delay decreases. Starting phase of the first steering curve clock, must be set to 4 ("100") for dynamic receive operation. For normal operation (not in dynamic receive), the 'StartPhase' parameter can be used to increase the sampling resolution to 5 ns.	
. .	. .	. .	. .	
<58:56> <59> <62:60> <63>	3 1 3 1	Multiplier<7> MultiSign<7> StartPhase<7> Reserved	Multiplication factor of the master curve When '1' the delay increases, when '0' the delay decreases. Starting phase of the first steering curve clock, must be set to 4 ("100") for dynamic receive operation.	
<67:64>	4	Duration<0>	Duration of the 1st segment of the 8-point piecewise linear curve.	2.56 µs
. .	. .	. .	. .	. .
<95:92>	4	Duration<7>	Duration of the 8th segment of the 8-point piecewise linear curve.	2.56 µs
<103:96>	8	Slope<0>	Slope of the 1st segment of the 8-point piecewise linear curve.	$\frac{5\ \text{ns}}{2.56\ \mu\text{s} \cdot 256}$
. .	. .	. .	. .	. .
<159:152>	8	Slope<7>	Slope of the 8th segment of the 8-point piecewise linear curve.	$\frac{5\ \text{ns}}{2.56\ \mu\text{s} \cdot 256}$
<161:160>	2	RandomOpt	Dithering of starting moment of the steering curves in the Y-direction: 00 : no additional time added 01 : between 0-1 x $5\ ns/64$ delay time added 10 : between 0-3 x $5\ ns/64$ delay time added 11 : between 0-7 x $5\ ns/64$ delay time added	
<167:162>	6	Reserved	-	

7.1.6. REG_CERXTXCONF

The ASIC is an array of 64x16 elements. The REG_CERXTXCONF register can be utilized to apply quick aperture changes.
Note that REG_CERXTXCONF can only be used in the combination with compressed delays (REG_CONFIG.BeamRunRxCalc=1; REG_CONFIG.BeamRunTxCalc=1). For non-compressed delays, one could set Tx delays (TXDELAY_DFF) to -1 and Rx delays (RXDELAY_DFF) to 0 to disable the elements. Note that - if not using the driver - you can set Tx delays (TXDELAY_DFF) to 31 to disable elements and Rx delays (RXDELAY_DFF) to 0 to disable elements.
To save RX power, the group needs to be disabled (CONFIG_DFF.ODRV_EN=0, CONFIG_DFF.BIAS_EN=0).

Due to a silicon bug (Chapter 10), this register does not work as intended.
Leave it on the default values (<..>Min=0, RX_XMax=63, RX_YMax=15, TX_XMax=63, TX_YMax=15).
Set CONFIG_DFF.CW_EN=0 and use the register CWSEL_DFF (Section 7.2.5) instead to disable elements.

Table 21: REG_CERXTXCONF.

Address	# Bits	Name	Description
<5:0>	6	RX_XMin	Elements that fall within these boundaries (rectangle) are enabled for receive.
<13:8>	6	RX_XMax	
<19:16>	4	RX_YMin	
<27:24>	4	RX_YMax	
<37:32>	6	TX_XMin	Elements that fall within these boundaries (rectangle) are enabled for transmit.
<45:40>	6	TX_XMax	
<51:48>	4	TX_YMin	
<59:56>	4	TX_YMax	
<63>	1	TX_AppByVal	Instead of enabling elements by specifying the boundaries (rectangle), it is also possible to enable them by their delay value. Only elements with a delay between TX_MinVal and TX_MaxVal will be enabled. Due to a silicon bug, this functionality is not available, see the Silicon Errata in Chapter 10 for more information.
<39:32>	8	TX_MinVal	The minimum delay for which elements are enabled when enabled by value.
<47:40>	8	TX_MaxVal	The maximum delay for which elements are enabled when enabled by value. Due to the same silicon bug mentioned above, this points to the same address as TX_MinVal <39:32>.

7.1.7. REG_CONFIG

Table 22: REG_CONFIG.

Address	# Bits	Name	Description
<7:0>	8	TxClkDivider	Transmit clock divider, see digital modules Factional 4.4 bit values, examples: 16 = 50MHz (20ns) 20 = 40MHz (25ns) 24 = 33MHz (30ns) 28 = 28.5MHz (35ns) 32 = 25MHz (40ns) Minimum value is 16, maximum value is 255.
<10:8>	3	TXDELAY_SHIFT	Shift in Y-direction for the transmit delay calculation to compensate for the position of the ASIC when multiple ASICs are used in an array: 1 ASIC : $ASIC_1$="011" 2 ASICs : $ASIC_1$="010" $ASIC_2$="100" 3 ASICs : $ASIC_1$="001" $ASIC_2$="011" $ASIC_3$="101" 4 ASICs : $ASIC_1$="000" $ASIC_2$="010" $ASIC_3$="100" $ASIC_4$="110" The middle of the complete array is always based at value "011". (See Figure 8 for a clarification.)
<12:11>	2	TXDELAY_LSBOPT	LSB Handling for the transmit delay calculation (for debugging purposes): 00 : normal 01 : SKIPLSB_0 (set LSB to 0) 10 : SKIPLSB_1 (set LSB to 1) 11 : INLSB (invert LSB)
<14:13>	2	TXDELAY_LSBRAND	LSB Randomization: 00 : no randomization 01 : quarter LSB randomization 10 : half LSB randomization 11 : full LSB randomization
<15>	1	RXDELAY_OFFSET	If set to '1', an offset of 1 LSB is applied to the element delay calculation assignment scaler. This is used to utilize 7 or 8 steering curves for dynamic receive.
<18:16>	3	RXDELAY_SHIFT	Same as their transmit equivalents described above, but for receive.
<20:19>	2	RXDELAY_LSBOPT	
<23:21>	3	RXDELAY_LSBRAND	
<24>	1	UseAnalogReset	Use Analog Reset at Beam Start
<25>	1	RandomizeDynUpdate	Randomize LSB for dynamic update
<26>	1	BeamRunRxCalc	Run the receive delay calculations when triggered
<27>	1	BeamRunTxCalc	Run the transmit delay calculations when triggered
<28>	1	BeamRxClkInit	Run µBeamformer Initialization when 'SetupRxTime' has elapsed
<29>	1	BeamContRx	Continue the receive clock after 'StopRxTime' has elapsed
<30>	1	BeamContTx	Continue the transmit clock after 'StopTxTime' has elapsed
<31>	1	BeamWaitRun	Wait for 'TriggerRun' to start beam, use 'TriggerPagexx' for beam preparation
<39:32>	8	Error_NMask_Out	Bitmask to mask errors seen on the external QSPI_ERROR pin.
<47:40>	8	Error_NMask_XMit	Bitmask to mask errors that automatically stop transmit.
<48>	1	LockDuringBeam	Makes all commands except for the 'TriggerStop' command invalid during transmit and receive.

Address	# Bits	Name	Description
<52:49>	4	Error_Stopmask	When transmit is stopped because of an error condition, automatically stop the following core modules by resetting all configuration registers: ---1 : Output driver periphery --1- : Receive state-machine -1-- : Transmit state-machine 1--- : All bias configuration The cleared registers would have to be re-configured after error handling.
<53>	1	-	Reserved
<55:54>	2	TxClk_Config	Bit 0: unused Bit 1: If set to '1' the transmit clock generation circuit only uses the rising edges of the incoming clock to generate the transmit clock instead of using both its rising and falling edge.

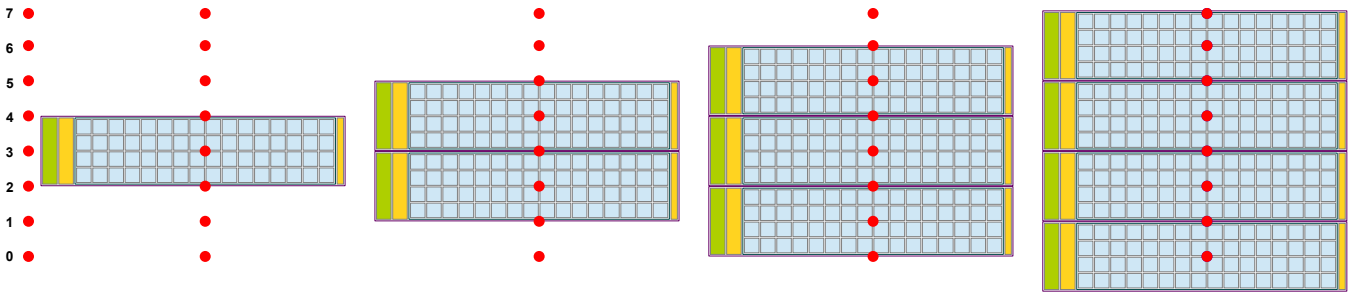


Figure 8: TXDELAY_SHIFT clarification.

7.2. Core

Core registers (all have the suffix "_DFF") are group-level registers that exist locally in every group and contain group- and element-level configuration data. Therefore, there are 64 instances of all the registers listed in Table 23. Note for sending the chain of 64 core registers, a dummy byte is needed at the end (before the CRC is calculated).

Table 23: Core Registers.

Register	Length in bits	Description
SELECT_DFF	2	Core chain selection for addressing which registers are updated when issuing the WriteChainCond* commands.
TXDELAY_DFF	88	Transmit delays for all 16 elements and a common group delay (offset).
RXDELAY_DFF	128	Receive delays for all 16 elements and a common dynamic group delay (offset) for near field imaging.
MATRIX_DFF	238	Memory of Read and Write pointers.
CWSEL_DFF	64	Continuous-Wave mode channel selection per element.
XMITWF_DFF	64	Transmit waveform setup (e.g. number of pulses, pulse-width, polarity).
CONFIG_DFF	40	Analog group configuration (e.g. currents, enable, power saving options).

** WriteChain is hex code 6x, where x=0..7. WriteChainCond is hex code Cx. For every WriteChain a corresponding WriteChainCond command exists (except ChainSELECT).*

As can be seen in Figure 9, the output of every group in a row is connected to the input of the next group in the row. During normal setup, the incoming data is shifted through (from left to right) until they arrive at the correct position. This means that changing a setting for one of the groups in a row, requires all other groups in the same row to be written as well. See Appendix A.1 and Appendix A.2 for code examples for scrambling and descrambling.

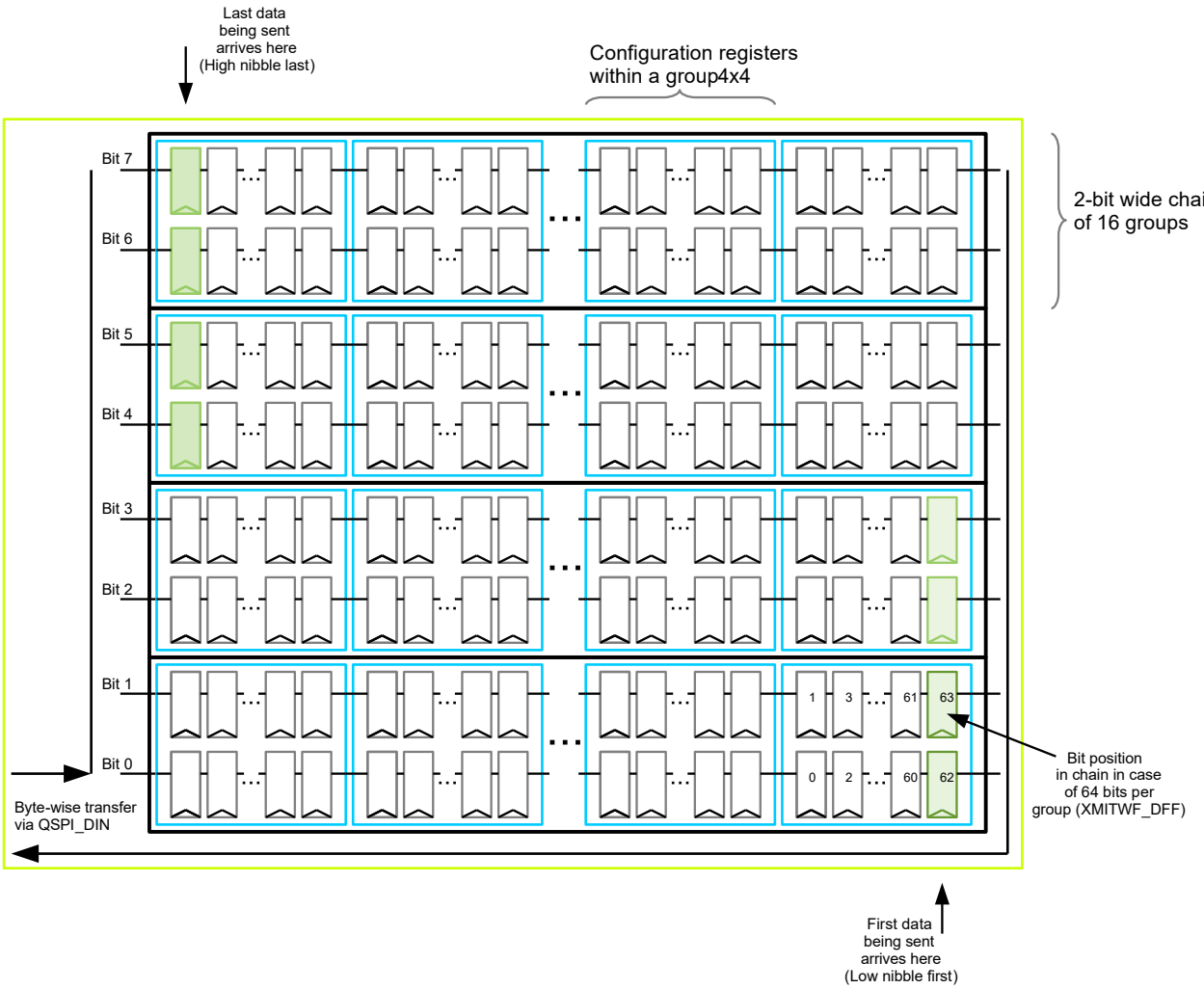


Figure 9: Core register data structure.

7.2.1. SELECT_DFF

Table 24: SELECT_DFF.

Address	# Bits	Name	Description
<1:0>	2	WriteCondSelect	'X0': WriteChainXXX, normal writing mode '01': WriteChainCondXXX, conditional writing mode: don't take data for this group '11': WriteChainCondXXX, conditional writing mode: take data for this group See also Figure 9.

It is important to note that when SELECT_DFF is used, the data length is reduced and should be adjusted accordingly.

7.2.2. TXDELAY_DFF

Table 25: TXDELAY_DFF.

Address	# Bits	Name	Description	Range
<4:0>	5	TX_ELEMENTDELAY<0>	TX Element 0 delay	The delays are in units of $0.5 \cdot TX_{CLK}$ and are calculated by: $delay = N \cdot \frac{TX_{CLK}}{2}$; $N = 0, \ldots, 29$ Element is disabled by setting: $N = 30, 31$ The start offset is in units of whole clock cycles and is calculated by: $offset = N \cdot TX_{CLK}$; $N = 0, \ldots, 255$
<9:5>	5	TX_ELEMENTDELAY<1>		
<14:10>	5	TX_ELEMENTDELAY<2>		
<19:15>	5	TX_ELEMENTDELAY<3>		
<24:20>	5	TX_ELEMENTDELAY<4>		
<29:25>	5	TX_ELEMENTDELAY<5>		
<34:30>	5	TX_ELEMENTDELAY<6>		
<39:35>	5	TX_ELEMENTDELAY<7>	TX Element 7 delay	
<44:40>	5	TX_ELEMENTDELAY<8>		
<49:45>	5	TX_ELEMENTDELAY<9>		
<54:50>	5	TX_ELEMENTDELAY<10>		
<59:55>	5	TX_ELEMENTDELAY<11>		
<64:60>	5	TX_ELEMENTDELAY<12>		
<69:65>	5	TX_ELEMENTDELAY<13>		
<74:70>	5	TX_ELEMENTDELAY<14>		
<79:75>	5	TX_ELEMENTDELAY<15>	TX Element 15 delay	
<87:80>	8	TX_GROUP_STARTCOUNTER	Start offset for all group elements	

7.2.3. RXDELAY_DFF

Table 26: RXDELAY_DFF.

Address	# Bits	Name	Description	Range
<4:0> <7:5>	5 3	RX_ELEMENTDELAY<0> RX_ELEMENT_CLKSEL<0>	RX Element 0 delay RX El 0 dynamic curve assign	The delays are in units of $0.5 \cdot RX_{CLK}$ and are calculated by: $delay = N \cdot \frac{RX_{CLK}}{2}$; $N = 3, \ldots, 27$ where RX_{CLK} is 25 MHz and the delay is in units of 20 ns. Receive is disabled by setting: $N < 3$ $N > 27$
<12:8> <15:13>	5 3	RX_ELEMENTDELAY<1> RX_ELEMENT_CLKSEL<1>		
<20:16> <23:21>	5 3	RX_ELEMENTDELAY<2> RX_ELEMENT_CLKSEL<2>		
<28:24> <31:29>	5 3	RX_ELEMENTDELAY<3> RX_ELEMENT_CLKSEL<3>		
<36:32> <39:37>	5 3	RX_ELEMENTDELAY<4> RX_ELEMENT_CLKSEL<4>		
<44:40> <47:45>	5 3	RX_ELEMENTDELAY<5> RX_ELEMENT_CLKSEL<5>		
<52:48> <55:53>	5 3	RX_ELEMENTDELAY<6> RX_ELEMENT_CLKSEL<6>		
<60:56> <63:61>	5 3	RX_ELEMENTDELAY<7> RX_ELEMENT_CLKSEL<7>	RX Element 7 delay RX El 7 dynamic curve assign	
<68:64> <71:69>	5 3	RX_ELEMENTDELAY<8> RX_ELEMENT_CLKSEL<8>		
<76:72> <79:77>	5 3	RX_ELEMENTDELAY<9> RX_ELEMENT_CLKSEL<9>		
<84:80> <87:85>	5 3	RX_ELEMENTDELAY<10> RX_ELEMENT_CLKSEL<10>		
<92:88> <95:93>	5 3	RX_ELEMENTDELAY<11> RX_ELEMENT_CLKSEL<11>		
<100:96> <103:101>	5 3	RX_ELEMENTDELAY<12> RX_ELEMENT_CLKSEL<12>		
<108:104> <111:109>	5 3	RX_ELEMENTDELAY<13> RX_ELEMENT_CLKSEL<13>		
<116:112> <119:117>	5 3	RX_ELEMENTDELAY<14> RX_ELEMENT_CLKSEL<14>		
<124:120> <127:125>	5 3	RX_ELEMENTDELAY<15> RX_ELEMENT_CLKSEL<15>	RX Element 15 delay RX El 15 dynamic curve assign	

7.2.4. MATRIX_DFF

A one-hot encoded translation of the RXDELAY_DFF register.

7.2.5. CWSEL_DFF

Table 27: CWSEL_DFF (if CONFIG_DFF.CW_EN=1).

Address	# Bits	Name	Description
<1:0>	2	CW_SEL_ELE<0>	CW channel selection bits in CW mode when TX_GND_CW = '0' and TX_HVP_CW = '0': 00 : channel 0 01 : channel 1 10 : channel 2 11 : channel 3
<2> <3>	1 1	TX_GND_CW<0> TX_HVP_CW<0>	CW drive selection bits in CW mode: 00 : normal CW channel (see above) 01 : connected to GND 10 : connected to HVP 11 : no connection, high-impedance
. . . .	. . . .	. . . .	. . . .
<61:60>	2	CW_SEL_ELE<15>	CW channel selection bits in CW mode when TX_GND_CW = '0' and TX_HVP_CW = '0': 00 : channel 0 01 : channel 1 10 : channel 2 11 : channel 3
<62> <63>	1 1	TX_GND_CW<15> TX_HVP_CW<15>	CW drive selection bits in CW mode: 00 : normal CW channel (see above) 01 : connected to GND 10 : connected to HVP 11 : no connection, high-impedance

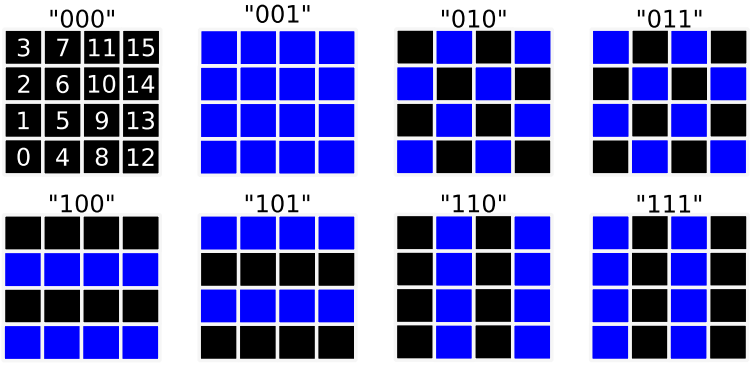
Table 28: CWSEL_DFF (if CONFIG_DFF.CW_EN=0[4]).

Address	# Bits	Name	Description
<1:0>	2	CW_SEL_ELE<0>	<0> Disable RX <1> Disable TX
<3:2>	2	-	Reserved
. . . .	. . . .	. . . .	. . . .
<61:60>	2	CW_SEL_ELE<15>	<0> Disable RX <1> Disable TX
<63:62>	2	-	Reserved

[4] Receive or transmit can only be disabled if CONFIG_DFF.DISABLE_DISABLE=0.

7.2.6. XMITWF_DFF

Table 29: XMITWF_DFF Register.

Address	# Bits	Name	Description
<3:0>	4	TX_STARTSAMPLE	Pointer to the first bit of the waveform head.
<7:4>	4	TX_REPSAMPLE	Pointer to the first bit of the waveform body.
<11:8>	4	TX_ENDSAMPLE	Pointer to the last bit of the waveform body.
<13:12>	2	TX_PRECHARGE	Pre-charge timing control for waveforms starting in a high state: 00 : NO_PRECHARGE 01 : RANDOM (see also CONFIG_DFF.RANDOM_REG_UPD) 10 : FOCUSED 11 : PLANE
<19:14>	6	TX_PATTERNREP	Number of main waveform repetitions. If set to 63, the main waveform (body) will be repeated until short cmd TriggerStop. **Note that for setting 63, the power consumption on HVP in the ASIC is high and the ASIC could be damaged.**
<24:20>	5	TX_CLAMPTIME	Clamping-time after transmit in units of 4·TX_{CLK} periods. **TX_CLAMPTIME must be >=4.**
<28:25>	4	TX_PRECHARGETIME	***Only if TX_PRECHARGE is not RANDOM:*** Pre-charge time before transmit in units of 16·TX_{CLK} periods
<26:25>	2	TX_PRECHARGE_ RANDOM_CONF	***Only if TX_PRECHARGE is RANDOM:*** Specifies the range in which pre-charging should be enabled (in units of TX_{CLK}): Value / Precharge start moment / Total time (before+after start) '00' / 0—15 / 32 '01' / 0—31 / 64 '10' / 0—63 / 112 '11' / 0—63 / 160
<28:27>	2	TX_PRECHARGE_SLOPE	***If TX_PRECHARGE=RANDOM and TX_PRECHARGE_RANDOM_ISRC='1':*** $I_{PRECHARGE} = 1.52 \cdot I_{REF} \cdot ISEL\_PCHVP$, where ISEL_PCHVP for S-curve: Seg / Bit / ΔISEL_PCHVP for '0' / ΔISEL_PCHVP for '1' 0 / - / Start ISEL_PCHVP=0 / Start ISEL_PCHVP=0 1 / <27> / +1 / +2 2 / <28> / +2 / +3 3 / <28> / +2 / +3 4 / <28> / +2 / +3 5 / - / 0 / 0 6 / <28> / -2 / -3 7 / <28> / -2 / -3 8 / <28> / -2 / -3
<31:29>	3	TX_CHSEL_PATTERN	Waveform selection mask (Black = Waveform 0, Blue = Waveform 1).: 

<63:32>	32	TX_WF_SETUP	Two 16-bit waveform data sets are available per group. Even bits describe waveform 0, odd bits describe waveform 1. Every bit has a duration of 1·TX_{CLK} period. Each waveform consists of three parts: 1. The head of the waveform is executed first and is stored in bits `TX_WF_SETUP(TX_STARTSAMPLE : TX_REPSAMPLE-1)` 2. The body of the waveform is the main part of the waveform where all pulses should be stored and is stored in bits `TX_WF_SETUP(TX_REPSAMPLE : TX_ENDSAMPLE)` The body is executed 1 + TX_PATTERNREP times. 3. The tail of the waveform is executed last and is stored at bits: `TX_WF_SETUP(TX_ENDSAMPLE + 1 : 15)`
<39:32>	8	TGC_GAIN	Manual overwrite of the TGC gain step if CONFIG_DFF.RES_CNTL_OVERWRITE==1. Note that these bits are also part of the waveform setup; set TX_STARTSAMPLE>=4 to ignore these bits as part of the waveform. '00000000' -20 dB; '00000001' -17.5 dB; '00000011'-15 dB; '00000111' -12.5 dB; '00001111' -10 dB; '00011111' -7.5 dB; '00111111' -5 dB; '01111111' -2.5 dB; '11111111' 0 dB. **Note:** when TGC gain overwrite is used, do **not** resend XMITWF_DFF on every beam. This would cause artefacts during receive.

7.2.7. CONFIG_DFF

Table 30: CONFIG_DFF.

Address	# Bits	Name	Description
<3:0>	4	ISEL_LNA	LNA current[5]: $I_{LNA} = I_{REF} \cdot ISEL\_LNA$
<7:4>	4	ISEL_ODRV	Output-driver current: $I_{ODRV} = I_{REF} \cdot \frac{ISEL\_ODRV}{16} \cdot 20$
<11:8>	4	ISEL_RESCNTL	Controls the transition speed between TGC steps, has an insignificant current consumption.
<15:12>	4	ISEL_DCGND	Transmit discharge current: $I_{DISCHARGE} = 1.52 \cdot I_{REF} \cdot ISEL\_DCGND$ **Note:** ISEL_DCGND must be >=1
<19:16>	4	ISEL_PCHVP	Transmit precharge current: $I_{PRECHARGE} = 1.52 \cdot I_{REF} \cdot ISEL\_PCHVP$
<19:16>	4	ISEL_PCHVP	***If XMITWF_DFF.TX_PRECHARGE=RANDOM and CONFIG_DFF.TX_PRECHARGE_RANDOM_ISRC='1':*** Length of the segments of precharge S-curve: Val: '00' <17:16> segments 1,2,3,4,6,7,8: 3·TX_{CLK}; <19:18> segment 5: 4·TX_{CLK} '01': 7·TX_{CLK}; 8·TX_{CLK} '10': 11·TX_{CLK}; 12·TX_{CLK} '11': 15·TX_{CLK}; 16·TX_{CLK}
<20>	1	BIAS_EN	Group bias enable
<21>	1	ODRV_EN	Output driver core enable (high impedance when disabled; direct connection to the µBeamformer.)
<22>	1	RX_ALWAYS_EN	If '1', the analog T/R switch is always closed. The LNA is still protected, no risk of braking any circuitry, but it might have some leakage current and therefore reduced harmonic performance. Due to a silicon bug, RX_ALWAYS_EN must be set to 1. Alternatively, BeamContTx can be set to 1 at the cost of higher power consumption. **Note that if precharging is used, RX_ALWAYS_EN must be 0 and BeamContTx must be 1.** See the Silicon Errata in Chapter 10 for more information.

[5] I_{REF} = 10 µA.

<23>	1	CW_EN	Group CW enable If '0', transmit and receive can be disabled in the CWSEL_DFF register Table 28.
<24>	1	LNA_EN	LNA enable (when not in AUTO mode). Directly connected to the XDCR when disabled.
<25>	1	RES_CNTL_OVERWRITE	Manual overwrite of the TGC gain step in XMITWF_DFF<39:32>
<28:26>	3	RES_CAL	LNA bias resistor calibration current (I_OP). Sets operating point of LNA from 450 mV to 700 mV. Effects harmonic performance. Best value: 0. (compared to RES_CAL=5, Noise reduces from 50.4 µV to 49.9 µV, THD and HD2 reduce by 4 dB. Bandwidth is not affected. Operating points are not affected)
<29>	1	ANALOG_RESET_ATNOTRX	Automatic analog reset when not in RX/TX Due to a silicon bug, ANALOG_RESET_ATNOTRX must be set to 0.
<30>	1	-	Reserved
<31>	1	LNA_AUTO_POWERDOWN	Automatic LNA power-down when not in RX
<32>	1	TX_PRECHARGE_RANDOM_ISRC	When '0' $I_{PRECHARGE}$ is used as the precharge current. When '1' an S-shaped precharge ramp (which contains 7 segments) is generated, only available in random precharge mode (XMITWF_DFF.TX_PRECHARGE='01'). See Figure 10.
<33>	1	RANDOM_REG_UPD	When '0': generates a new pseudo-random number at the first transmit pulse only. Note that the seed based on Tx delays; so if all delays are 0 at the first pulse -> no random! When '1': generates a new pseudo-random number whenever a transmit pulse is sent.
<37:34>	4	MATRIX_OFFSET_VAL	The dither value between 0 -13. When 15, a pseudo-random dither value between 0 -13 is generated. Due to a silicon bug, the pseudo-random generator (value 15) does not work correctly because it generates values between 0-14. See the Silicon Errata in Chapter 10 for more information.
<38>	1	PCHVP_PS_EN	Enables power saving for the precharge operation.
<39>	1	DISABLE_DISABLE	If '1', transmit and receive can no longer be disabled in the CWSEL_DFF register Table 28.

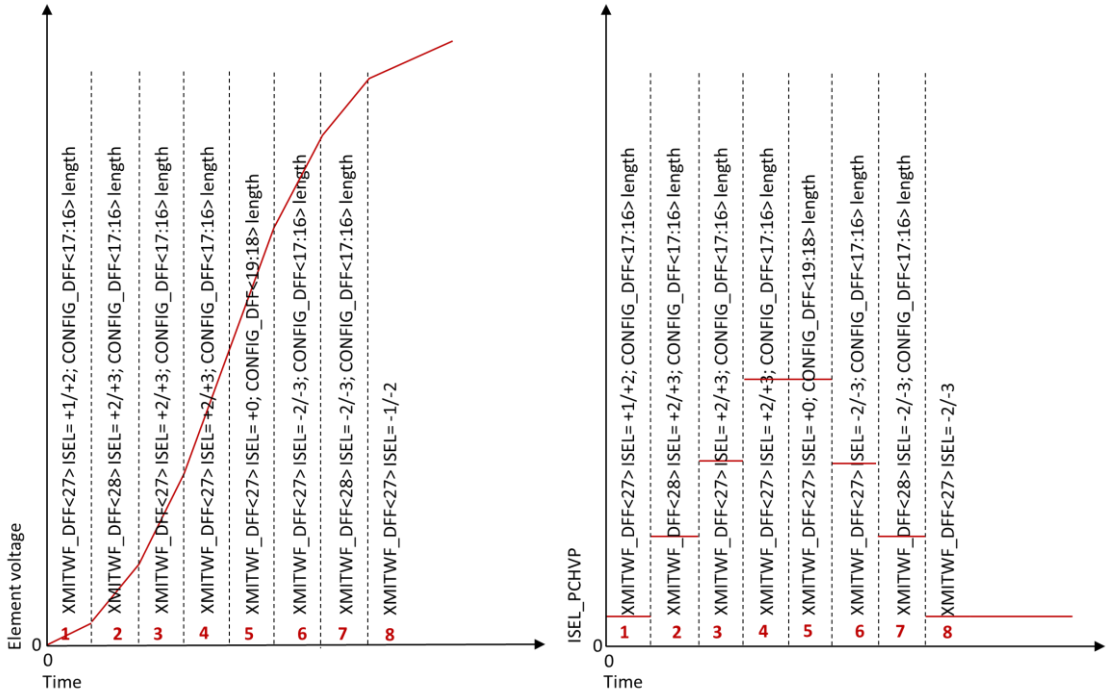


Figure 10. ISEL_PCHVP and element voltage for S-shaped precharge curve (TX_PRECHARGE_RANDOM_ISRC='1').

7.3. Output Driver

Each of the 64 output drivers has one set of the CONFIG_DRV register shown in Table 31 containing configuration data for the output drivers.

Table 31: Output Driver Registers.

Register	Length in bits	Description
CONFIG_DRV	16	Output driver configuration (e.g. currents, enable, input source selection)

7.3.1. CONFIG_DRV

Table 32: Output Driver Configuration Register (CONFIG_DRV).

Address	# Bits	Name	Description
<0>	1	DRV_ENABLE	Output driver periphery enable
<3:1>	3	DRV_BIASSEL	Output driver periphery current: $I_{ODRV,FF} = I_{REF} \cdot \frac{74}{1+DRV_{BIASSEL}} + 65\mu A$ $I_{ODRV,FB} = I_{REF} \cdot \frac{95}{1+DRV_{BIASSEL}} + 65\mu A$
<7:4>	4	DRV_SELINCW	CW-channel input source selection (one-hot encoded): 0000 : not connected 0001 : channel 0 0010 : channel 1 0100 : channel 2 1000 : channel 3
<11:8>	4	DRV_SELOUTCW	Connect driver cable to CW_CH<3:0> 0000 : not connected 0001 : channel 0 0010 : channel 1 0100 : channel 2 1000 : channel 3
<12>	1	DRV_OUTCON	Connect driver cable to driver output
<13>	1	DRV_SELINNOCW	Set the µBeamformer output as the driver input
<14>	1	DRV_RES_EN	Connect bias resistors to the driver input (100 kΩ to V_{DD}, 35 kΩ to GND)
<15>	1	DRV_FF_nFB	If set to '1' the output driver periphery will operate in feedforward mode If set to '0' the output driver periphery will operate in feedback mode

The order in which the output driver data is stored is defined by the layout of the output drivers in the periphery. The CONFIG_DRV registers are configured as a chain that connects all groups as shown in Figure 11. See Appendix A.3 and Appendix A.4 for code examples of scrambling and descrambling.

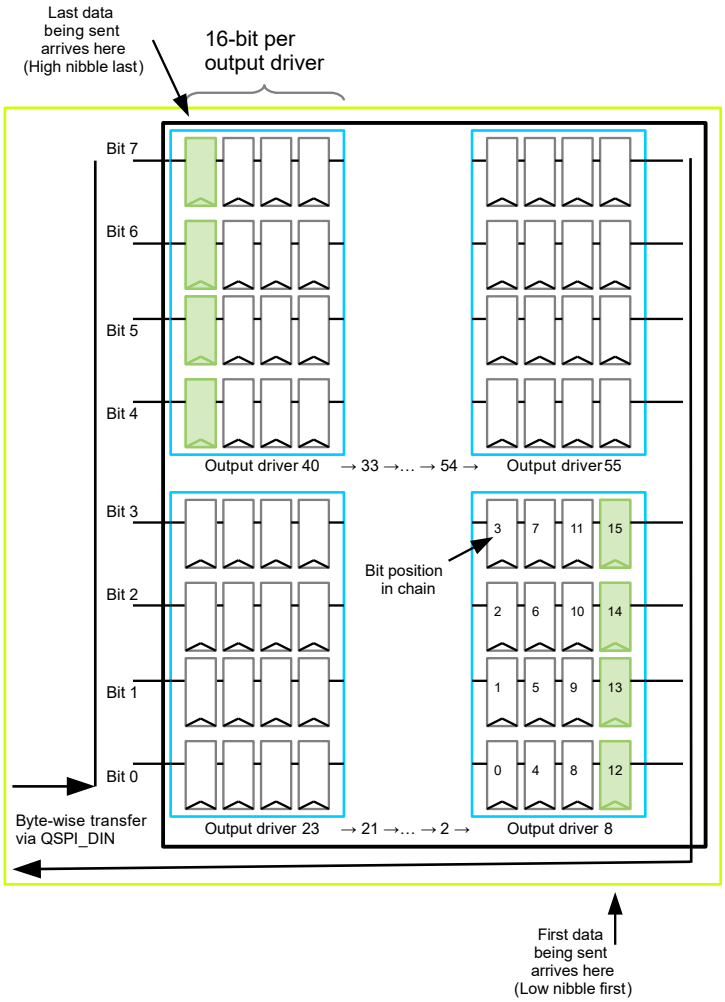


Figure 11: Output driver data structure.

The data order is shown below in Table 33 where the data shifts from left to right. I.e. data for groups 8 and 55 is written first and is shifted trough the complete structure.

Table 33: Output driver group data order.

40	33	32	42	41	43	34	36	35	45	44	46	37	39	38	56	47	57	48	50	49	59	58	60	51	53	52	62	61	63	54	55
23	21	31	30	29	20	22	18	28	27	26	17	19	16	25	24	15	6	7	4	14	13	12	3	5	1	11	10	9	0	2	8

8. Module Descriptions and Performance

In this chapter all implemented modules and their performance will be described. Unless stated otherwise, the default simulation settings shown in Table 34 have been used to perform the simulations/measurements.

Table 34: Default Simulation Parameters.

Register	Name	Value	Notes
REG_CONFIG	(Table 22)		
	TxClkDivider	20	40 MHz → 25 ns TX_{CLK} period
CONFIG_DFF	(Table 30)		
	ISEL_LNA	4	30 µA
	ISEL_ODRV	4	50 µA
	ISEL_RESCNTL	8	Insignificant
	ISEL_DCGND	8	160 µA
	ISEL_PCHVP	8	160 µA
	RES_CAL	5	550 mV
CONFIG_DRV	(Table 32)		
	DRV_BIASSEL	3	260 µA
	DRV_FF_nFB	0	Feedback mode
Other			
	C_{LOAD}	200p	Output driver loaded by a 200 pF capacitance

8.1. Overall US^X-ASIC Performance

In this section the most important performance parameters are summarized to give an overview of the ASIC performance.

Table 35: ASIC performance summary.

Symbol	Parameter	Value	Unit
$ASIC_{fc}$	ASIC cutoff frequency	7.2	MHz
$ASIC_{Vout,n}$	ASIC output referred noise	50	μV_{RMS}
$ASIC_{NF}$	ASIC noise figure	4	dB
$ASIC_{DR}$	ASIC dynamic gain range	18	dB
$ASIC_{Vpp}$	ASIC output swing	750	mV
$ASIC_{THD,3M}$	ASIC total harmonic distortion w.r.t 3 MHz signal	38	dB
$ASIC_{PSRR}$	ASIC power supply rejection ratio	45	dB

The performance and power consumption of the receive circuitry of the ASIC is affected most by the following three parameters:

- ISEL_LNA
- ISEL_ODRV
- DRV_BIASSEL

Their effect on the noise, bandwidth and distortion are shown in the following sections.

8.1.1. Noise

The total output referred noise integrated in a band from 2.25 MHz – 8.00 MHz is shown in the figures below.

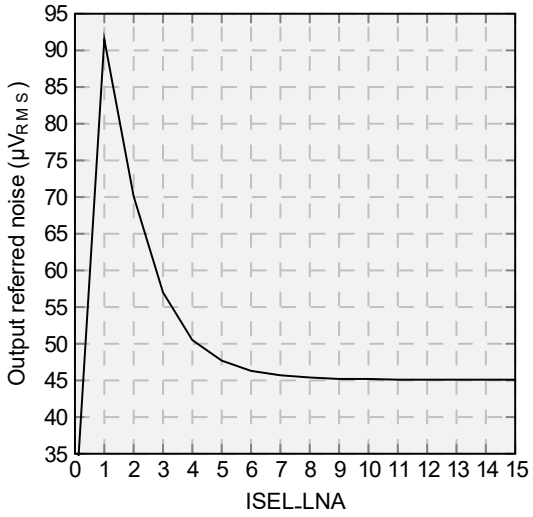


Figure 12: Output referred noise w.r.t. ISEL_LNA.

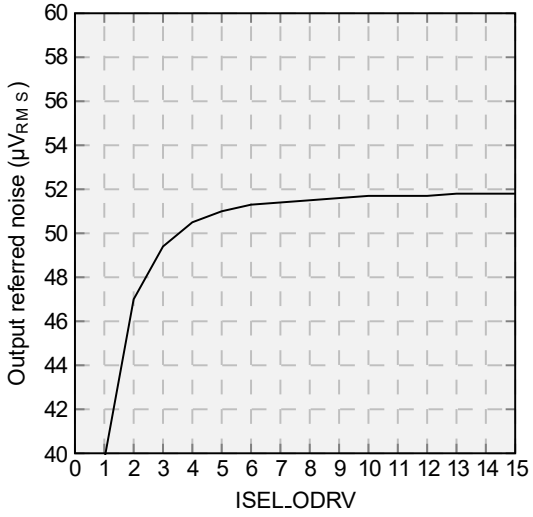


Figure 13: Output referred noise w.r.t. ISEL_ODRV.

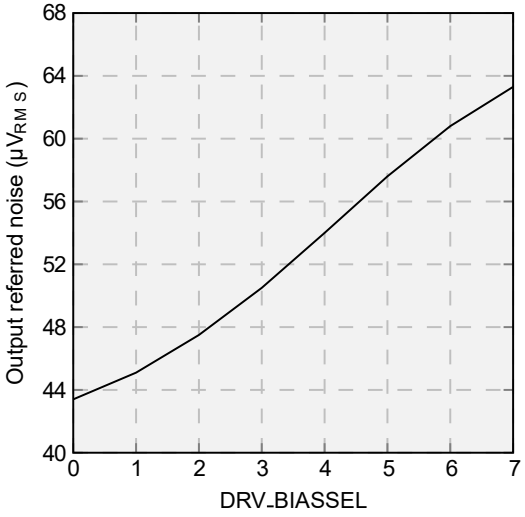


Figure 14: Output referred noise w.r.t. DRV_BIASSEL.

The distribution of the noise contributors at the output of the ASIC is summarized in the figure below.

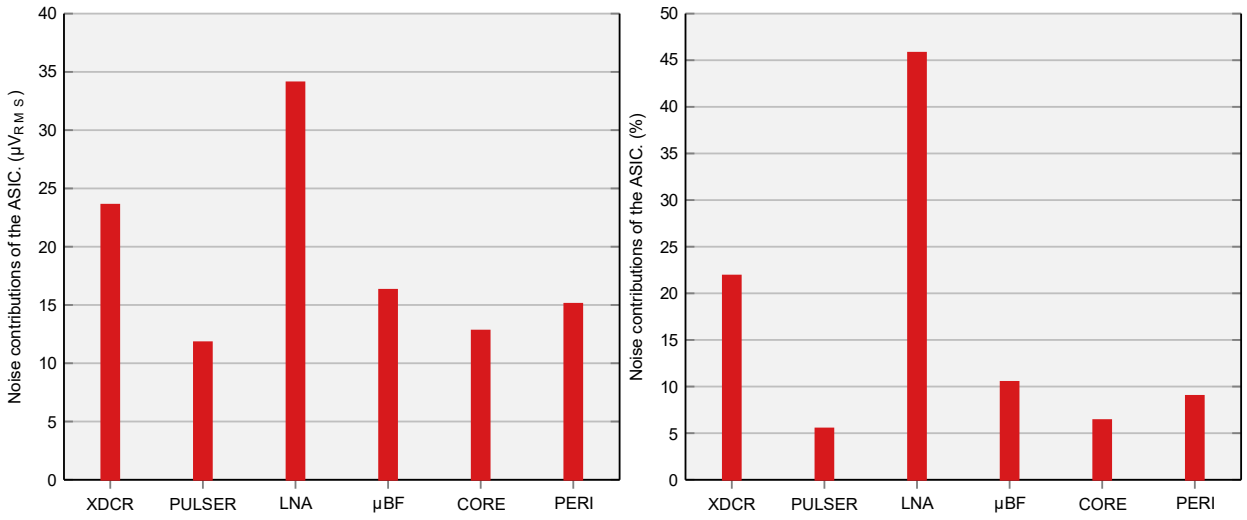


Figure 15: Output referred noise contributors including XDCR in μV_{RMS} (left) and percentages (right) integrated between 2.25 MHz – 8.00 MHz.

In the default settings, the ASIC has a noise figure of 4 dB, which corresponds to 50 μV_{RMS} at the output of the ASIC integrated between 2.25 MHz – 8.00 MHz. The noise figure and value can be reduced to roughly 2.8 dB and 40 μV_{RMS} by increasing the currents. This affects thermal performance but could be useful in certain applications. The noise spectral densities are shown in Figure 16 below.

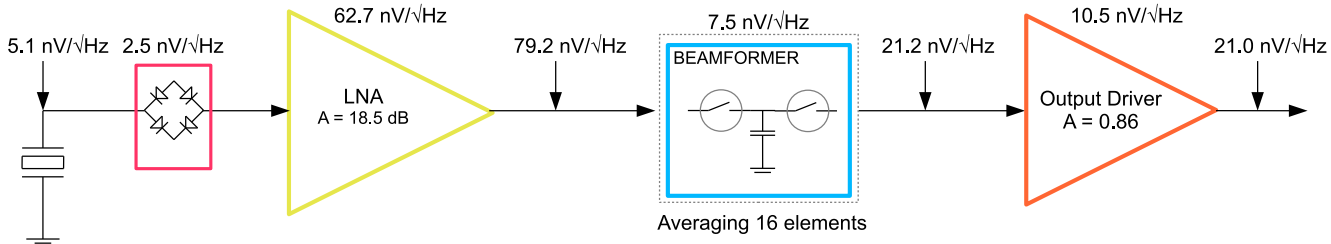


Figure 16: Noise spectral densities within a 2.25 MHz – 8.00 MHz band.

8.1.2. Bandwidth

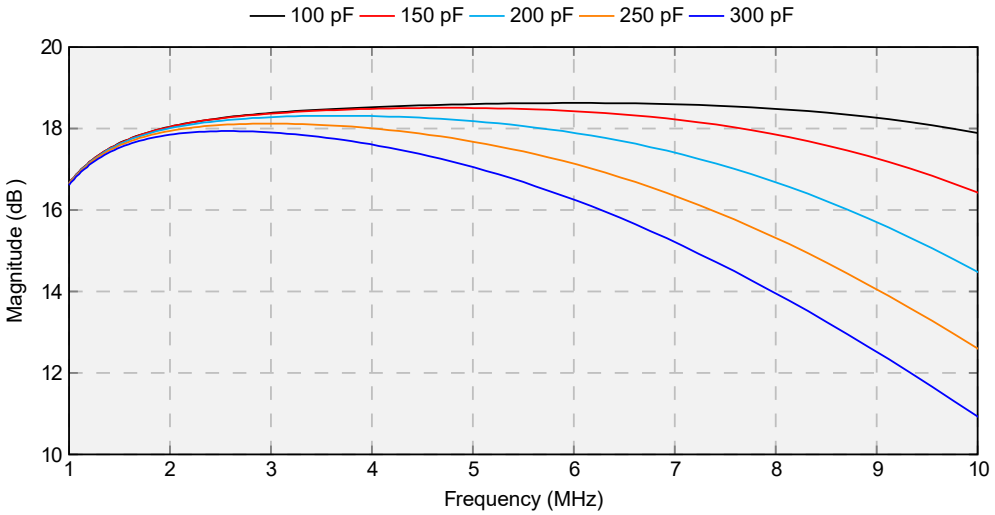


Figure 17: Bandwidth w.r.t. load capacitance C_{LOAD}.

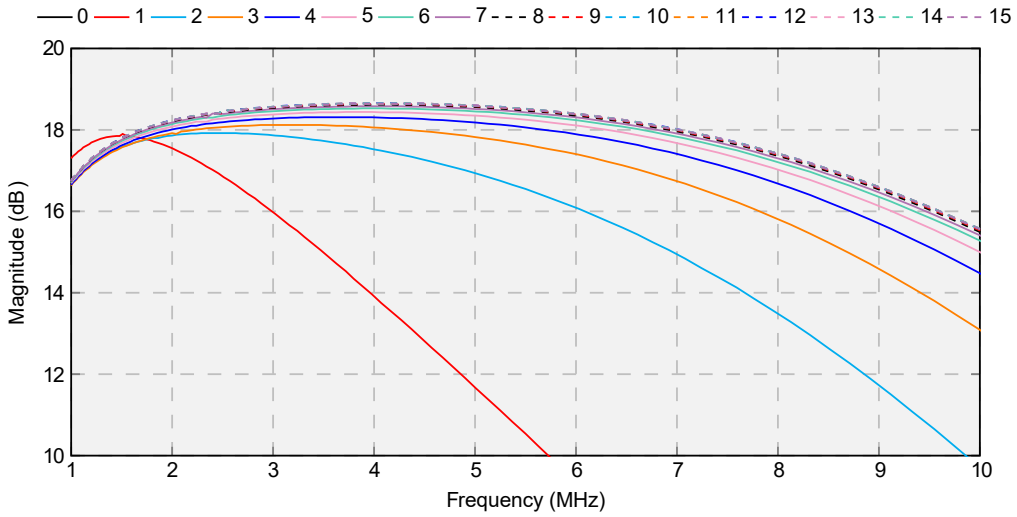


Figure 18: Bandwidth w.r.t. ISEL_LNA.

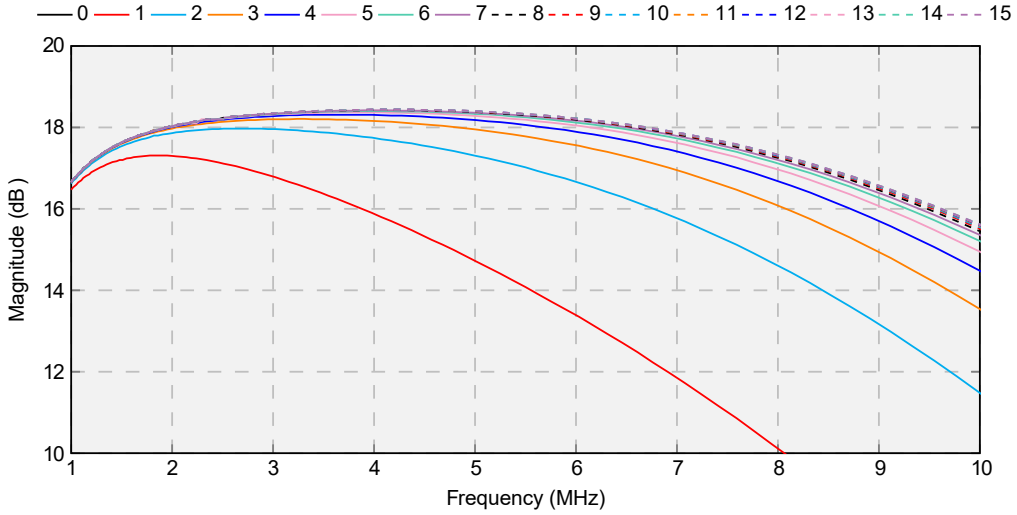


Figure 19: Bandwidth w.r.t. ISEL_ODRV.

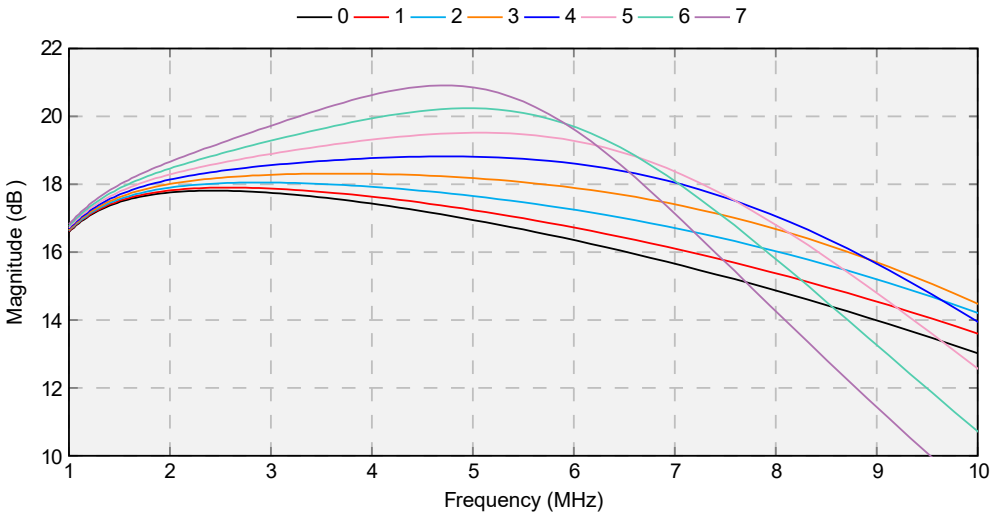


Figure 20: Bandwidth w.r.t. DRV_BIASSEL.

8.1.3. Distortion

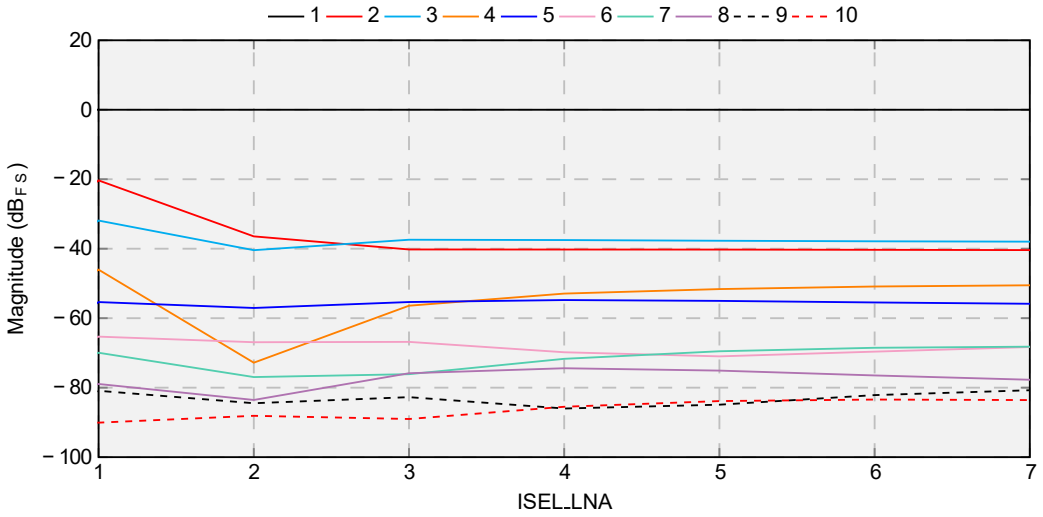


Figure 21: Harmonic distortion w.r.t. ISEL_LNA (3 MHz base tone).

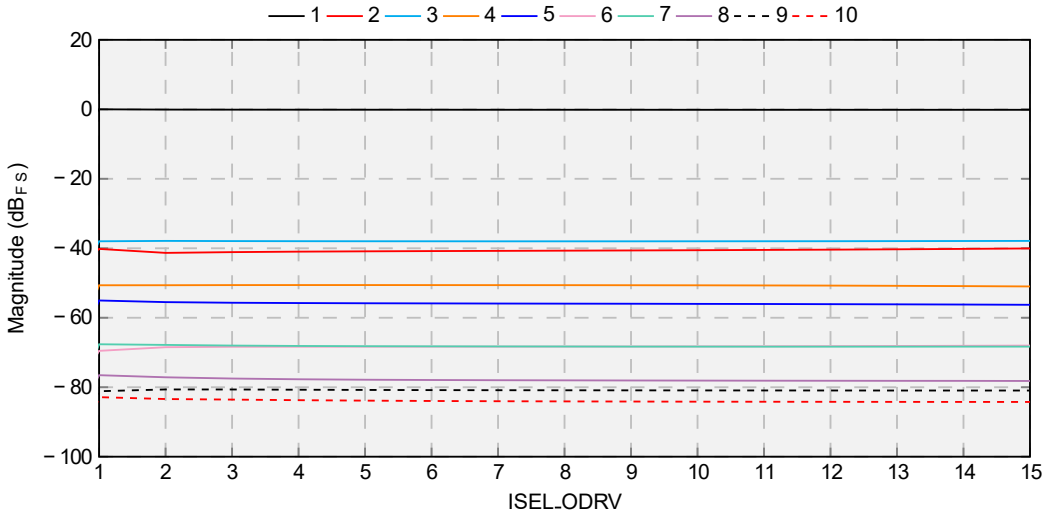


Figure 22: Harmonic distortion w.r.t. ISEL_ODRV (3 MHz base tone).

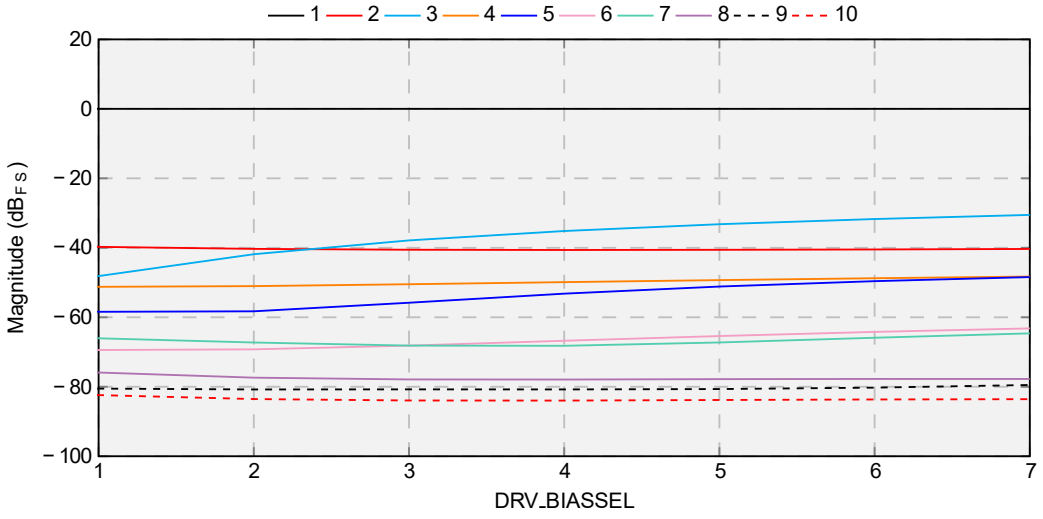


Figure 23: Harmonic distortion w.r.t. DRV_BIASSEL (3 MHz base tone).

8.1.4. Linearity

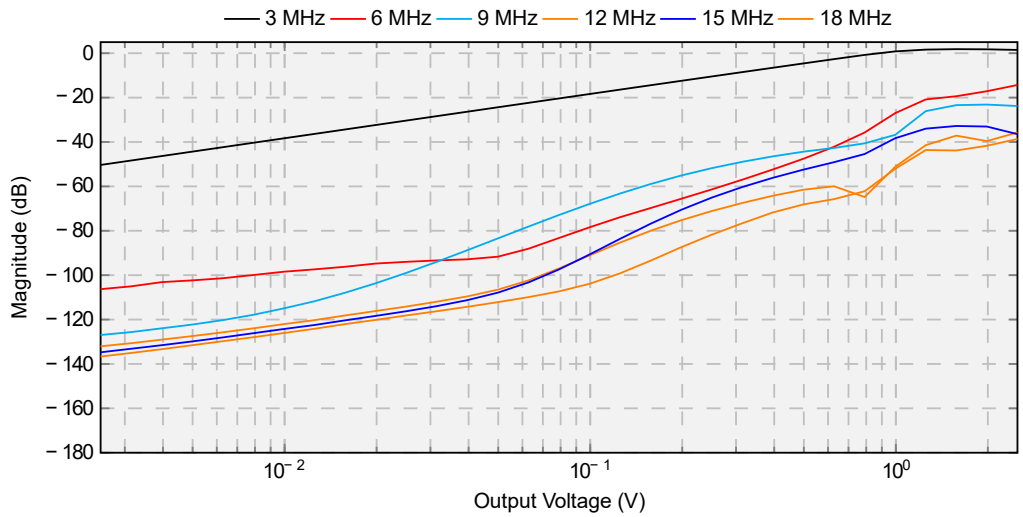


Figure 24: Harmonics of a 3 MHz base tone w.r.t. output amplitude.

8.1.5. Power Supply Rejection Ratio (PSRR)

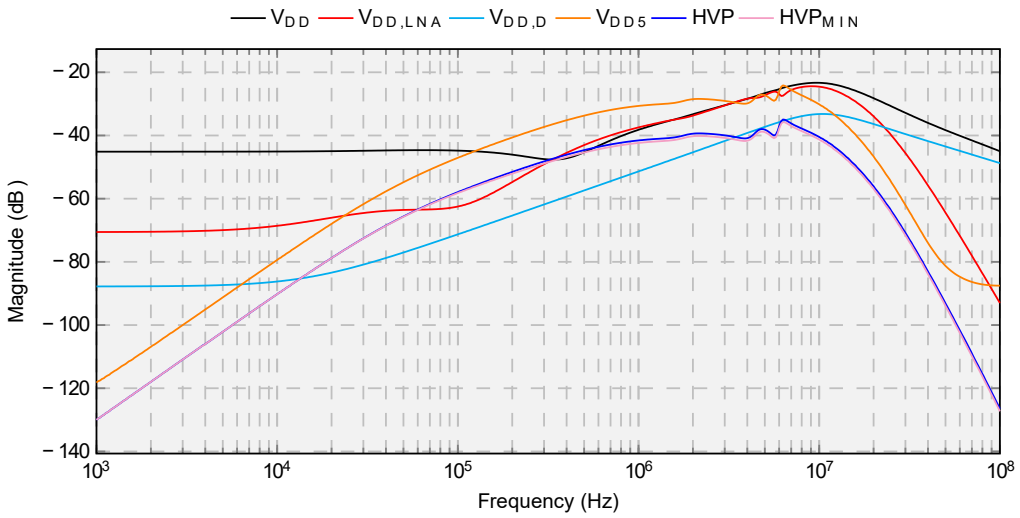


Figure 25: PSRR wr.t. the used voltage sources.

8.2. Core Modules

As described in Chapter 5, the core contains the following modules:

- 64 groups, consisting of:
  - 16 ‘Single Channels’ each containing:
    - Transducer pad to connect a transducer element
    - Two-level pulser for pulsing up to 100 V
    - Voltage limiter circuits to protect the receive circuitry from the high voltages
    - Time-gain compensation to attenuate strong near field signals
    - Low-noise amplifier signal gain
  - 1 µBeamformer for on-chip receive beamforming
  - 1 output driver for driving the combined signal to the periphery
  - Setup registers
- Waveform generator

A schematic overview of the core is repeated in Figure 26.

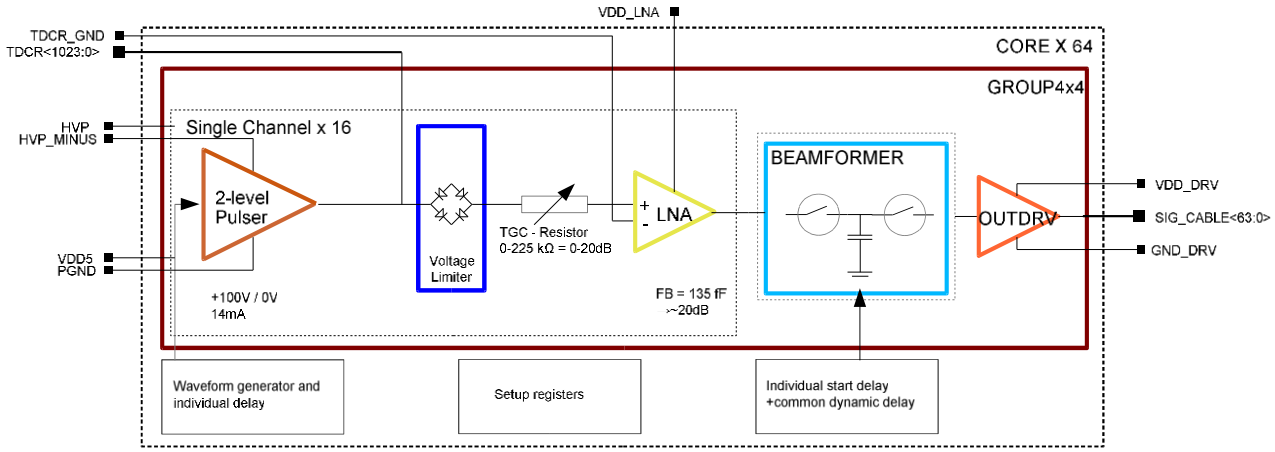


Figure 26: Schematic block diagram of ASIC core.

Descriptions and performance criteria of the core modules are given in the following subsections.

8.2.1. Two-Level Pulser

The two-level pulser is a high voltage driver capable of rapidly (dis)charging transducer elements with full drive strength or slowly (dis)charging them with programmable current sources. Slow transitions are acoustically invisible and can be used to charge the transducer up to HVP to generate a falling edge first. This can be used for pulse inversion. When operated in pulsed mode, the pulser is controlled by the waveform generator discussed in section 8.2.2. that generates the WG_+ and WG_- signals. A simplified schematic diagram of the pulser is shown in Figure 27.

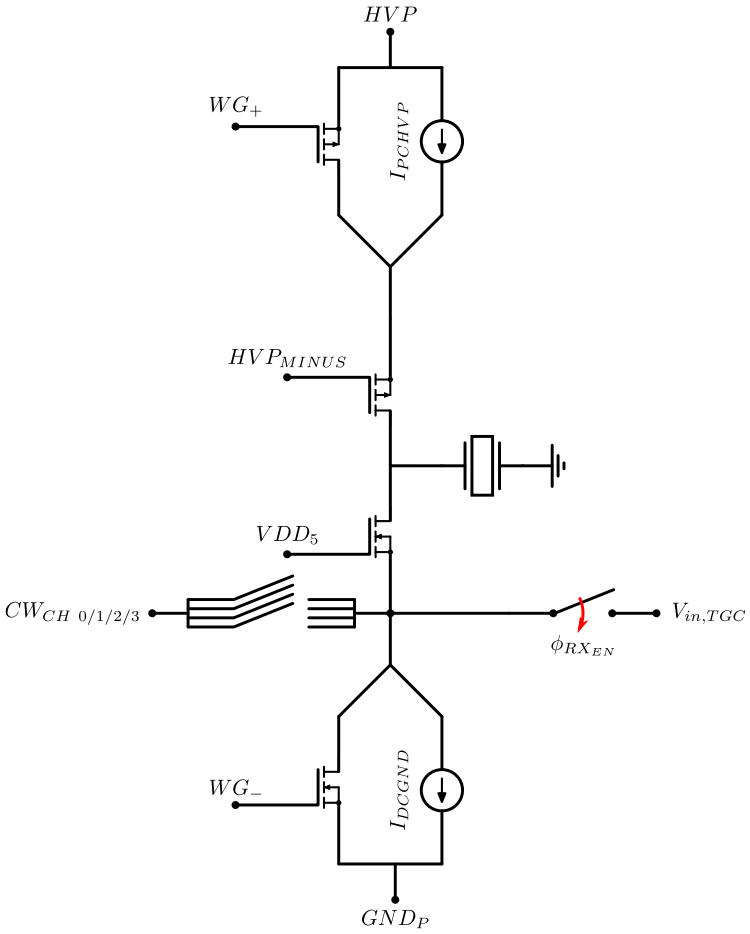


Figure 27: Simplified schematic diagram of the two-level pulser.

The receive circuitry is separated from the pulser during transmission by means of a T/R-switch. The T/R-switch limits the maximum voltage at the receive circuitry to 1.8 V and is opened during transmit and closed during receive.

The ASIC can also be operated in a continuous-wave (CW) mode by connecting the transducer elements to one of the 4 CW channels. The CW channels will be discussed in section 8.2.7 and CW mode will be discussed in section 9.7.

The most important design/performance parameters of the two-level pulser are summarized in Table 36 and the drive strength is shown in Figure 28.

Table 36: Electrical characteristics of the two-level pulser.

No.	Parameter	Condition	Symbol	MIN	TYP	MAX	UNIT
1	Drive strength high side	HVP 100 V, XDCR 50 V	TLP_STR_HS		15		mA
2	Drive strength low side	HVP 0 V, XDCR 50 V	TLP_STR_LS		15		mA
3	Rise-time	HVP = 100 V, 90% to 10%	TLP_FT		15		ns
4	Fall-time	HVP = 100 V, 10% to 90%	TLP_RT		15		ns
5	HD2	HVP = 50 V, CW-mode	TLP_HD2		-		dB
6	Pulse inversion	HVP = 50 V, 3 cycles, $dB20((V^+-V^-)/(V^++V^-))$	TLP_PI		-		dB

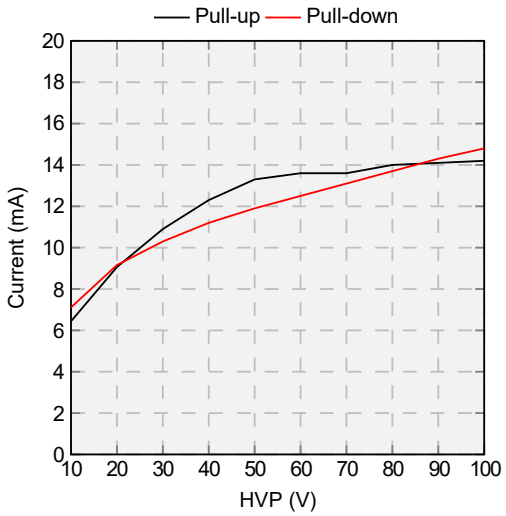


Figure 28: Drive strength of the two-level pulser.

8.2.2. Waveform Generator

Transmit waveforms are generated internally on the US^X-ASIC by the Finite State Machine (FSM) shown in Figure 29 on the next page. The FSM takes the parameters stored in the XMITWF_DFF register (Table 29) as an input to generate the desired waveforms. The FSM can be summarized in a few steps:

1. The pointer is adjusted to the TX_STARTSAMPLE which indicates the first bit of the described waveform.
2. The selected pre-charge option is executed.
3. The waveform is transmitted (head → body → tail)
4. The waveform is clamped if specified
5. The element starts receiving.

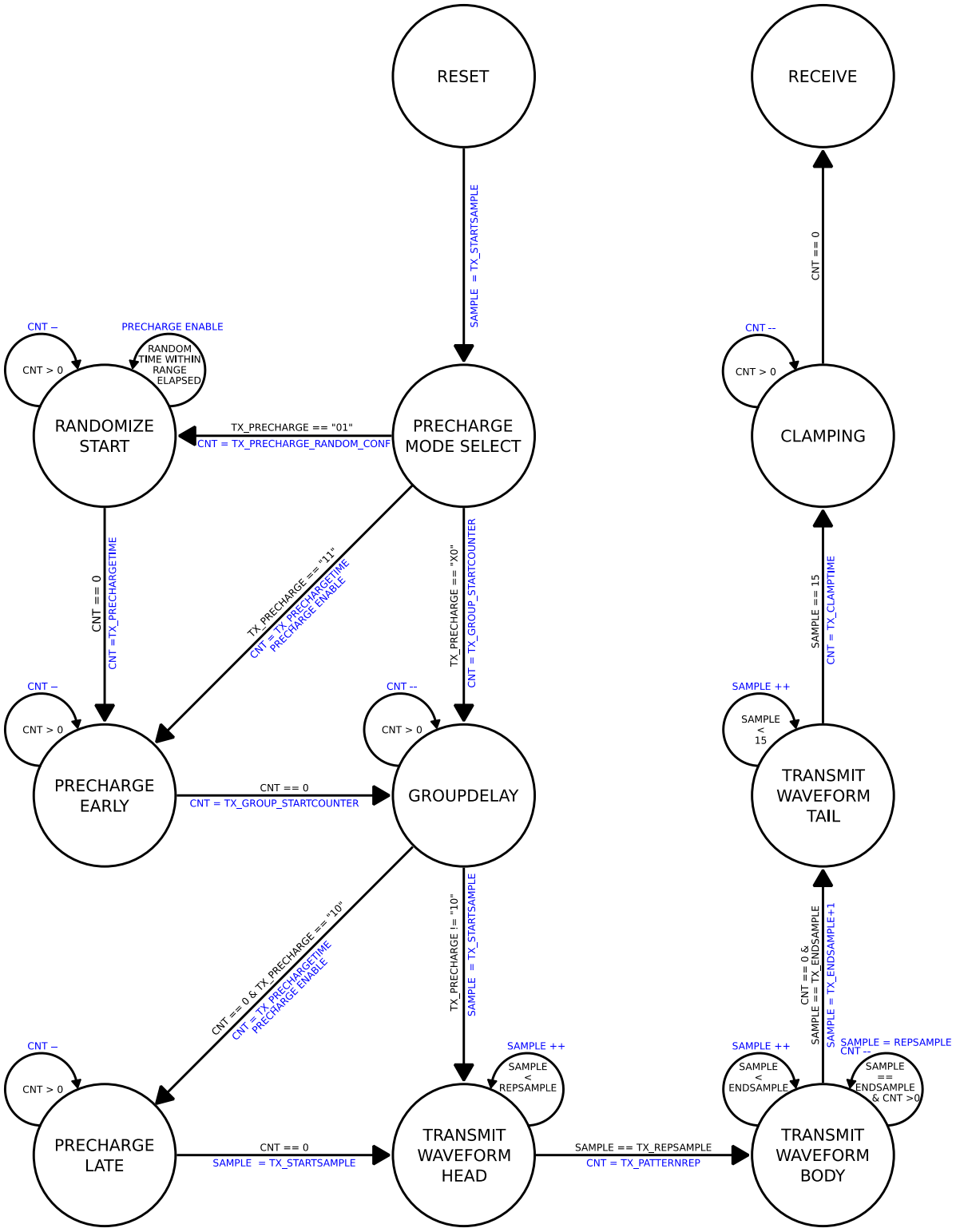


Figure 29: Waveform Generator FSM. (Condition, Assignment)

8.2.3. Time-Gain Compensation (TGC)

A switched resistor TGC scheme is implemented to decrease signal amplitudes in the near field that would otherwise lead to saturated output signals. When the receive phase starts, all switches are in an open state and the TGC is set to maximum attenuation. The maximum attenuation is 20 dB and decreases by 2.5 dB for every closed switch. The switches are closed slowly on an out-of-band frequency to prevent excitation of the transducer elements. The switching speed is controlled by the ISEL_RESCNTL parameter in the CONFIG_DFF register (Table 30). The REG_BEAMTIMING register (Table 16) is used to configure the rest of the TGC. A schematic representation of the TGC is given in Figure 30.

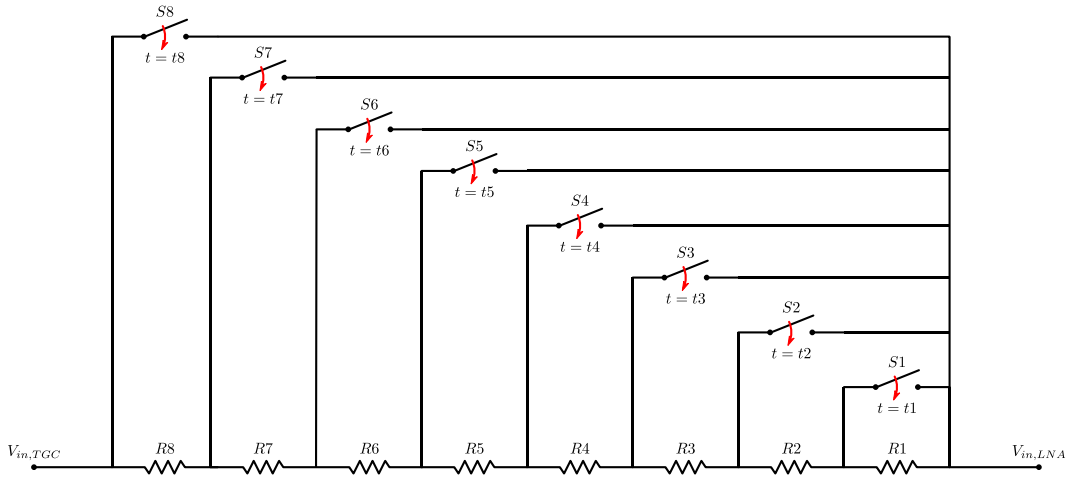


Figure 30: Simplified TGC schematic.

8.2.3.1. Charge Cancellation

Since the TGC is implemented with switched resistors, some charge is always injected into/from the channel when the switches change state. Switches 1 through 7 are relatively small because their on-resistance is not crucial. The final switch (S8) removes all extra resistances from the path making its on-resistance important. To decrease the on-resistance of S8 its size has been increased. The bigger size also means its channel is bigger and requires more charge to fully form. A charge cancellation scheme has been implemented that releases charge from the closed switches to form the channel of S8, therefore reducing the excitation of the transducer elements. The REG_BEAMTIMING register holds the configuration parameters.

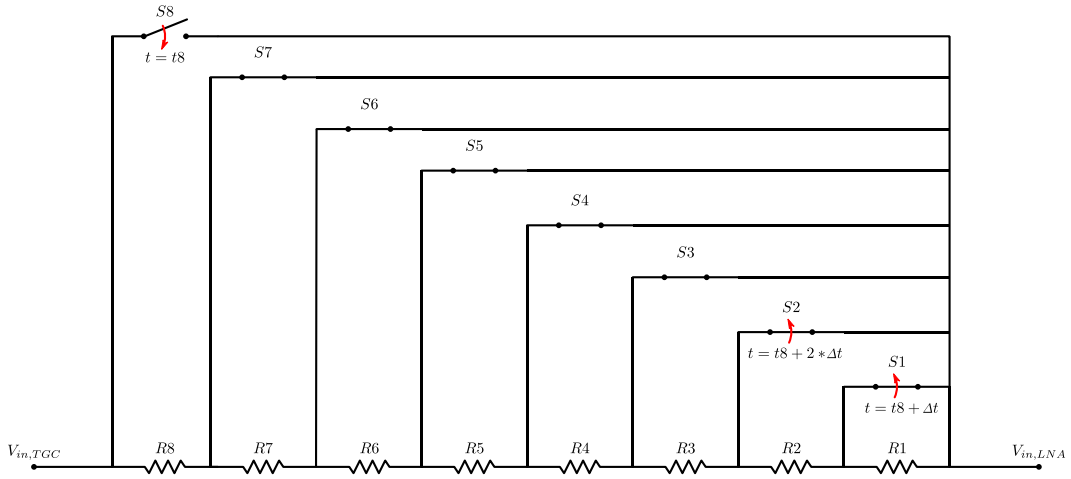


Figure 31: Simplified TGC charge cancellation scheme.

A small amount of charge is always injected into the channel as can be seen in Figure 32. Every spike corresponds to the closing of a TGC switch. The spikes are in the order of 5 μV_{rms} whereas the noise-level is 40 μV_{rms} so the switching should not be visible unless averaging is applied.

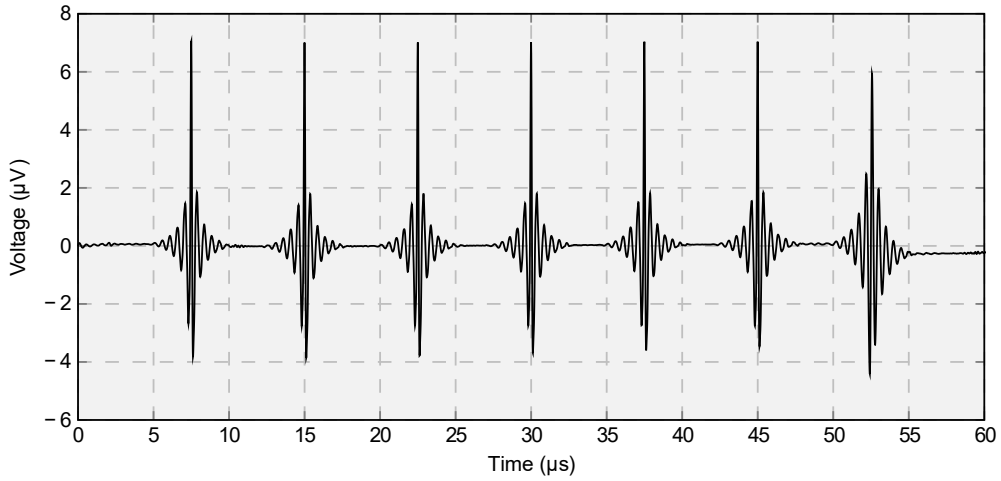


Figure 32: Channel charge injection due to TGC switching filtered between 2.25 – 8.00 MHz.

The bandwidth of the ASIC is reduced when the signal is attenuated. The result can be seen in the figure below, where 0 and 8 are respectively the lowest and highest gain settings. The reduced bandwidth will therefore only be visible in the near field where attenuation is applied.

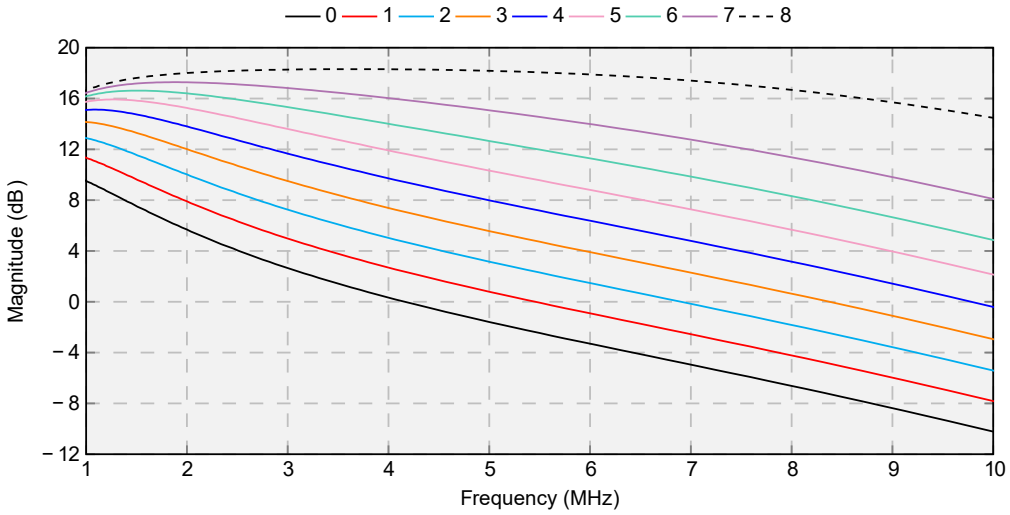


Figure 33: ASIC signal gain over frequency with respect to the TGC gain.

8.2.4. Low-Noise Amplifier (LNA)

The LNA is implemented as an AC-coupled two-stage amplifier with a fixed feedback loop. The first stage is a pseudo-differential common source amplifier with high gain for signal amplification and ground shift suppression. The second stage is a source follower implemented to drive the feedback loop with a constant current source load. The feedback loop consists of multiple branches:

- Anti-parallel diodes for oversteering protection
- A reset switch for fast settling after transmit
- A bypass switch to connect the input directly to the output when the LNA is powered down
- 1.35 pF input capacitor and 135 fF feedback capacitor for 10x signal gain
- 1 MΩ resistor for DC biasing and a lower corner frequency of 1 MHz

A simplified schematic representation of the LNA is shown in Figure 34.

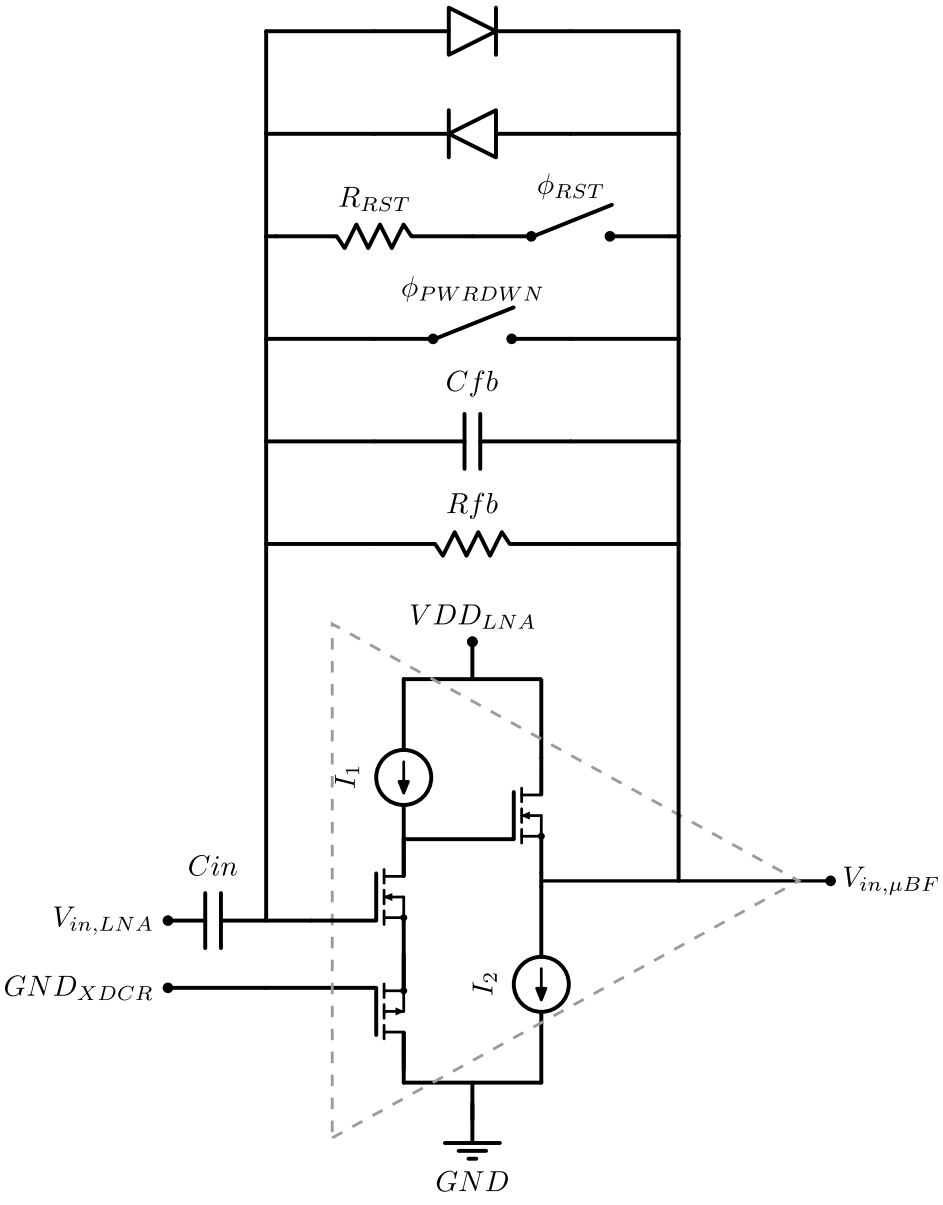


Figure 34: Simplified LNA schematic.

Table 37: Electrical characteristics of the LNA.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
LNA_A	LNA gain	@ 5MHz		18.5		dB
LNA_{fc}	LNA cutoff frequency			17.8		MHz
LNA_{Sn}	LNA noise spectral density	2.25 MHz – 8.00 MHz		62.65		nV/√Hz
LNA_{NF}	LNA noise figure			2.6		dB
LNA_{DR}	LNA dynamic gain range	Including TGC (20 dB)		83		dB
LNA_{Vpp}	LNA output swing			750		mV_{pp}
$LNA_{THD,3M}$	LNA THD w.r.t 3 MHz signal	750 mV_{pp} output swing, RES_CAL = 0		-43		dB

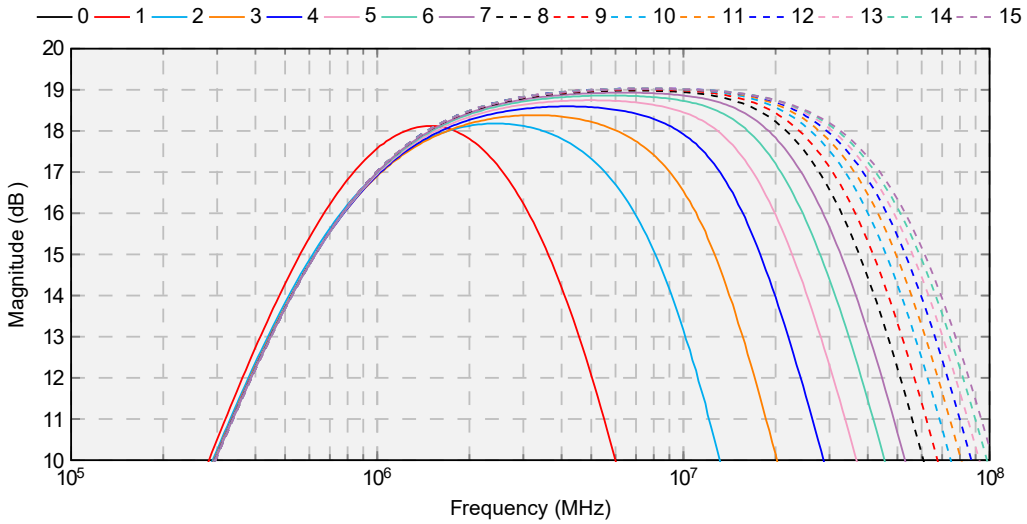


Figure 35: LNA bandwidth w.r.t. ISEL_LNA.

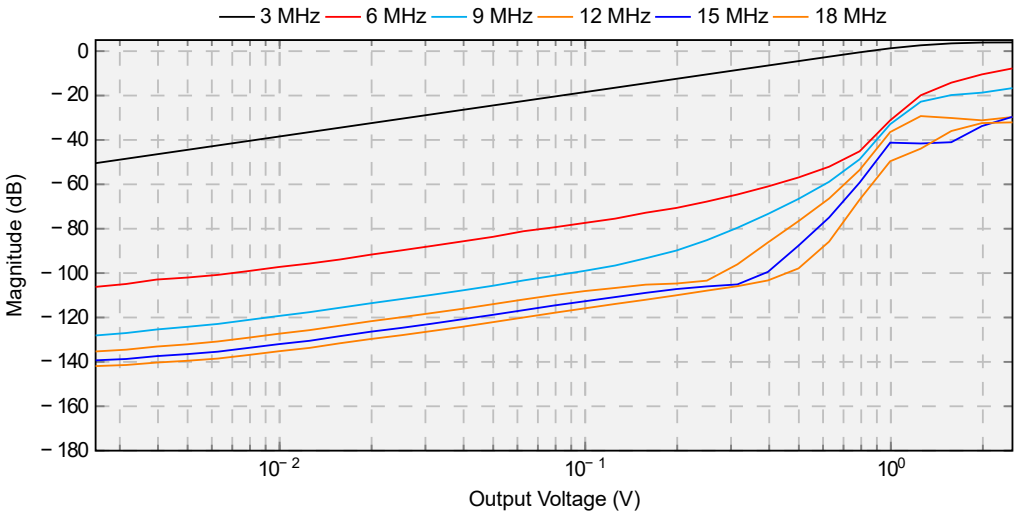


Figure 36: Harmonics of a 3 MHz base tone w.r.t. output amplitude.

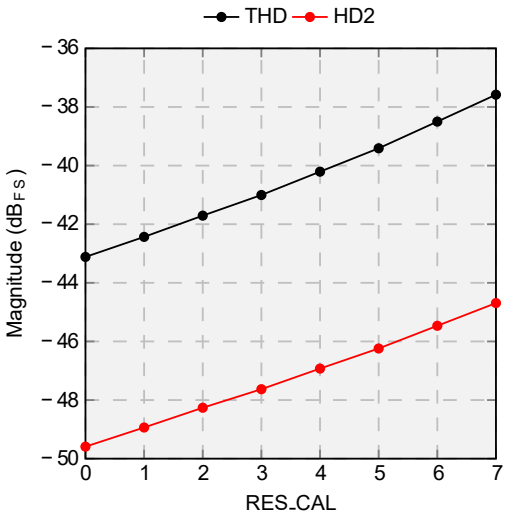


Figure 37: THD and HD2 w.r.t. RES_CAL (3 MHz base tone).

8.2.5. µBeamformer

The US^X-ASIC uses a µBeamformer to apply fine beamforming to elements within a group. The µBeamformer in each group contains 1 pipeline-operated S/H delay line per element. Each delay line consists of 14 capacitors that cyclically sample and hold the output of the LNAs at a frequency of 25 MHz (40 ns). The structure of the delay lines is shown in Figure 38.

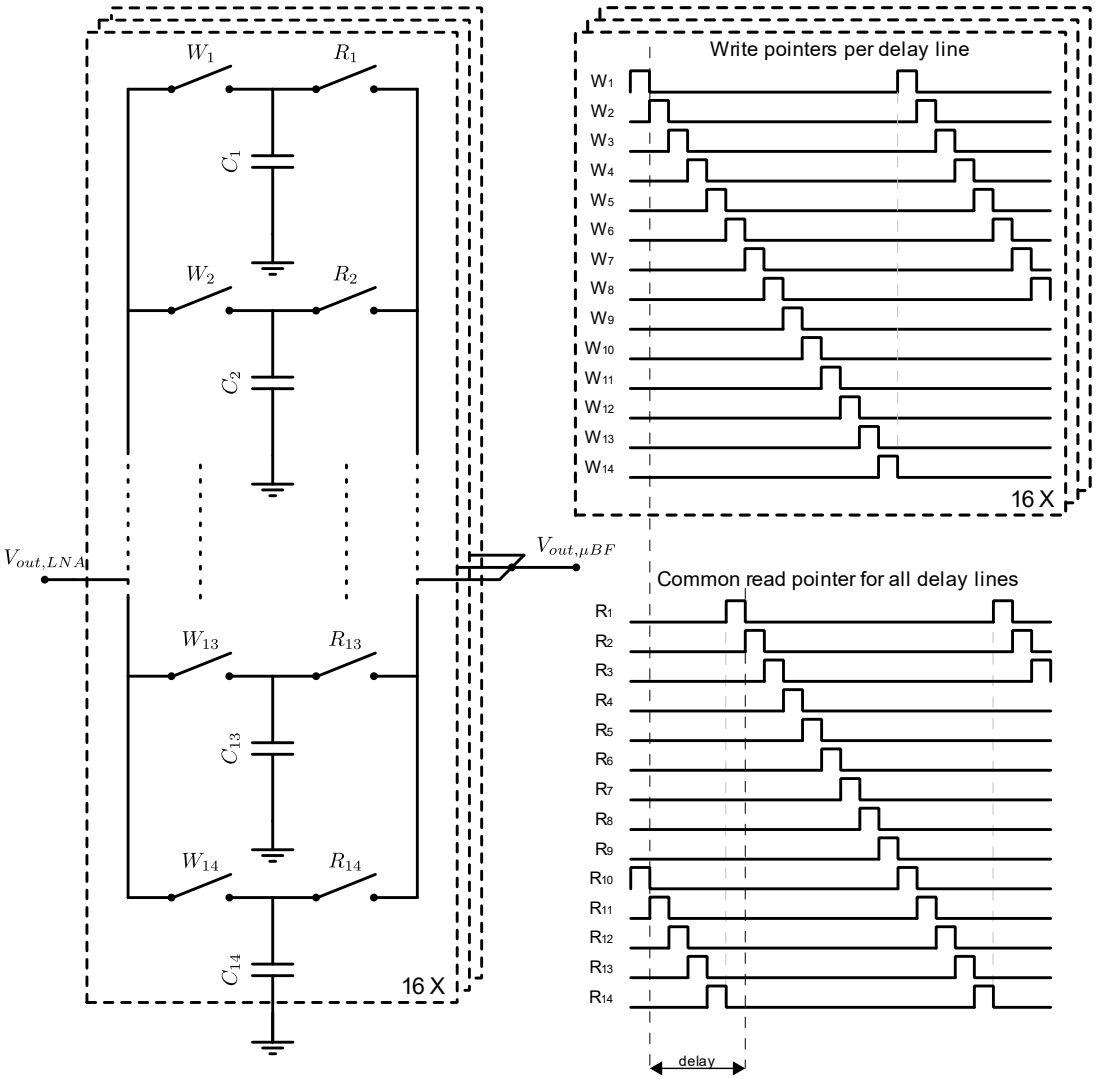


Figure 38:Pipeline operated S/H delay line.

The delays specified in the RXDELAY_DFF register are translated into write and read pointers for every delay line and are stored in the MATRIX_DFF register. Every group has 17 sets of the MATRIX_DFF register, where the first 16 registers point to the capacitor that is to be written first for every element. The 17th set is a shared read pointer that points to the capacitor that is to be read out. This capacitor is the same for all delay lines. Both write- and read pointers cycle through the 14 capacitors in steps of 40 ns. The difference in delays between the write pointers and the read pointer is what gives the beamforming. At readout, the charge in the capacitors is averaged into a single group output that is forwarded to the output driver. The delay-and-sum operation can be expressed by the following equation:

$$V_{in,ODRV_{CORE}} = V_{out,\mu BF} = \sum_{N=0}^{15} \frac{1}{16} S_N(t - \Delta T_N)$$

Where S_N represents the read out signal of the Nth element and ΔT_N represents the programmed delay for that element. The delay values can be programmed with a resolution of 20 ns and have a maximum delay of 480 ns as shown in Table 26.

8.2.5.1. Low pas filtering

Due to the sample-and-hold of 25 MHz, the receive signal is filtered by a sinc function with a notch at $f_{nyquist}$=12.5 MHz. The gain G as a function of frequency f is given by (Figure 39):

$$G(f) = \frac{\sin\left(\pi \cdot \frac{f}{f_{Nyquist}}\right)}{\pi \cdot \frac{f}{f_{nyquist}}}$$

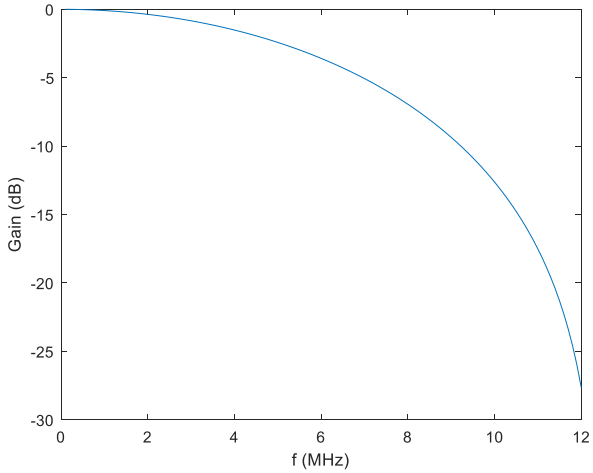


Figure 39: Gain for the Rx micro-beamformer (sampling frequency 25 MHz).

8.2.5.2. Dynamic Receive

The dynamic focusing functionality of the US^X-ASIC is achieved by changing the delays of the elements over time. This can be done by applying different frequencies to the write and read pointers. In practice, the substitution of only a few 40 ns cycles with 35 ns or 45 ns clock cycles is required.

A curve can be defined in the REG_RXDYN register that represents the change in delay over time. This so-called master curve M(t) is implemented as an 8-point piecewise linear function over time. From this master curve, 8 steering curves $C_{0...7}(t)$ can be generated that are multiples (fractions) of the master curve. Applying different scaling factors results in a different change of delay over time. Depending on the sign and slope of these curves, they result in the insertion of a number of 35 ns or 45 ns clock cycles into the delay lines.

A graphical is example is shown in Figure 40 for 5 elements moving from straight to tilted steering.

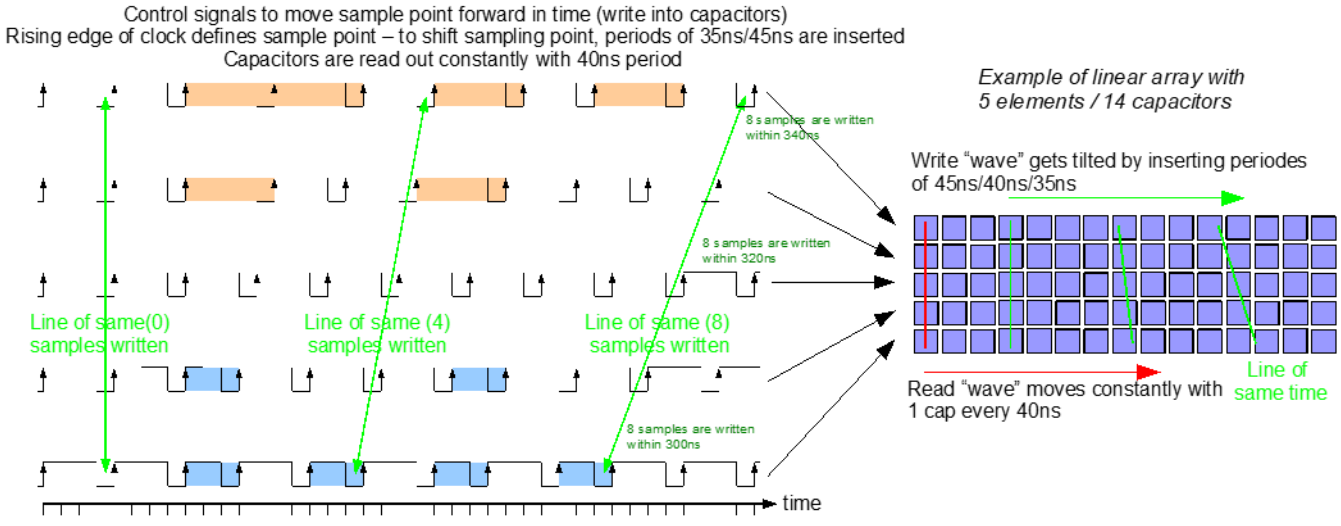


Figure 40: Dynamic steering example.

8.2.6. Output Driver Core

The output driver is split into two DC-coupled unity-gain stages:

- The output driver in the core used to buffer the µBeamformer output to the periphery
- The output driver in the periphery to drive the cable described in section 8.3.4

The output driver core is a simple differential amplifier with unity gain feedback as shown below.

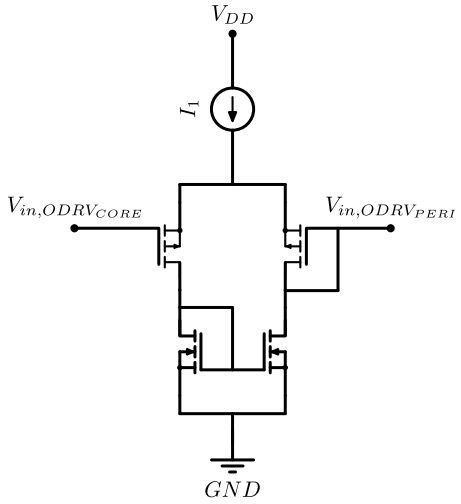


Figure 41: Simplified output driver core schematic.

The most important design/performance parameters are summarized in Table 38.

Table 38: $ODRV_{CORE}$ design/performance parameters.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
$ODRV_{CORE,A}$	Gain			0.89		V/V
$ODRV_{CORE,fc}$	Cutoff frequency			13.5		MHz
$ODRV_{CORE,DC,OP}$	DC operating point at in- and output			526		mV
$ODRV_{CORE,Vin,n}$	Noise at the input of $ODRV_{CORE}$	2.25 - 8.00 MHz		50.79		μV_{RMS}
$ODRV_{CORE,Sn}$	Noise Spectral Density	2.25 - 8.00 MHz		8.21		nV/√Hz
$ODRV_{CORE,NF}$	Noise figure			0.8		dB
$ODRV_{CORE,THD,3M}$	THD w.r.t. a 3 MHz tone	750 mV_{pp}		37		dB
$ODRV_{CORE,DR}$	Dynamic range	Including TGC (20 dB)		94.3		dB

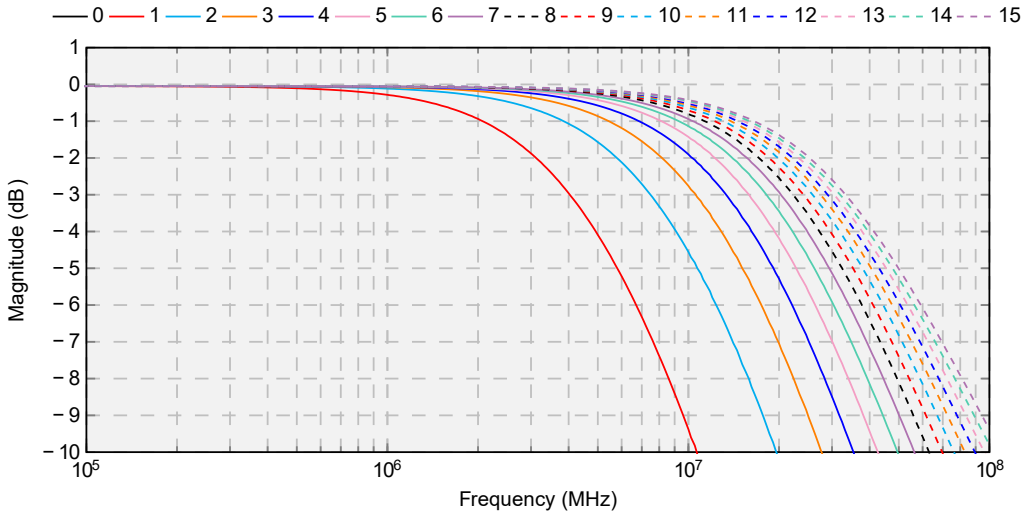


Figure 42: Output driver core bandwidth w.r.t. ISEL_ODRV.

8.2.7. Continuous-Wave Channels

Table 39: CW mode specification parameters.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
$R_{CW,ELE}$	CW mode switch resistance 512:4 multiplexer			300		Ω
$R_{CW,DRV}$	CW mode switch resistance 4:32 multiplexer			75		Ω

Table 40: CW mode maximum operating conditions.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
$V_{CW,MAX,TX}$	Maximum CW transmit voltage on CW_CH_TX<3:0>	OUT_CON must be set to '0' in the CONFIG_DRV register.	-0.5		10	V

8.3. Periphery Modules

As described in Chapter 5, the periphery contains the following modules:

- 64 output drivers to drive all 64 acoustic lines simultaneously
- Bias circuitry and calibration options
- Testing circuitry
- Fuse memory to write serial numbers
- Setup registers
- Communication interface
- Other digital blocks used for controlling the ASIC

A schematic overview of the periphery is shown in Figure 43. Descriptions and performance criteria of the periphery modules are given in the following subsections.

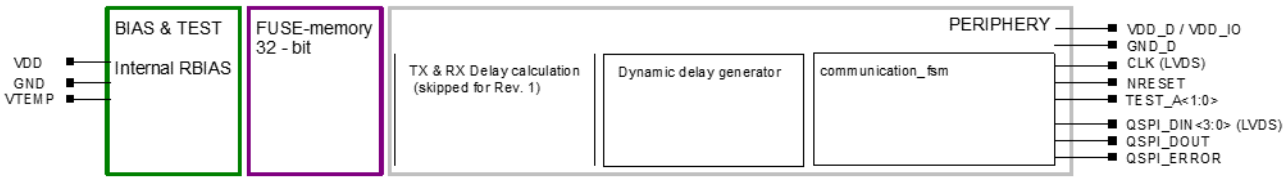


Figure 43: Schematic block diagram of the ASIC periphery.

8.3.1. Biasing and Analog References

The ASIC contains a temperature independent bandgap voltage reference V_{BG} that is used to provide:

- 10 µA current reference (I_{REF})
- Temperature dependent voltage (V_{TEMP})

Table 41: Biasing and reference specification.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
V_{BG}	Bandgap voltage			1.162		V
$I_{REF,NOCAL}$	Current reference	Uncalibrated	7	10	13	µA
I_{REF}	Current reference	Calibrated	9.8	10	10.2	µA
V_{TEMP}	Temperature sensor output voltage	300 K	0.6	0.9	1.2	V
S_{TEMP}	Temperature sensor sensitivity		22.0	22.4	22.8	mV/K

The temperature sensor has an output resistance of 640 kΩ. It is not possible to disable the output of the temperature sensor or alter its sensitivity with the current calibration. In order to enable the temperature sensor, V_{DD} must be applied. The sensitivity variation of the temperature sensor is negligible. The offset voltage however shows a greater variation and requires a one-point calibration at room temperature. All internal ASIC currents are referenced to (and thus also scale to) the current reference I_{REF}. The current should therefore be calibrated to the typical 10 µA value. The current value can be read out using the BIST ADC and can be adjusted by writing the fuse memory.

8.3.2. Fuses

The US^X-ASIC contains a 32-bit fuse memory. It is implemented using the XFAB PFUSE primitive devices in a 4-byte configuration. Table 42 lists the contents of the memory. Unblown fuses are read as ‘1’s, blown fuses are read as ‘0’s.

Table 42: Fuse memory.

Address	# Bits	Name	Description	Range
<31>	1	USED_FUSE	Used probe flag	0 – 1
<30:28>	3	REVISION_FUSE	ASIC revision number	1 – 7
<27:24>	4	IBIAS_CAL_FUSE	Reference current calibration value	0 – 15
<23:0>	24	SERIAL_FUSE	ASIC Serial Number: <23:15> Incremental Batch ID <14:10> Wafer ID <9:4> Chip location on wafer X <3:0> Chip location on wafer Y	 1 – 512 1 – 25 1 – 58 1 – 12

8.3.2.1. Reading Fuses

Fuses can be read using the BistGetFuseMemory command, returning all 32 bits of data.

8.3.2.2. Writing Fuses

In order to save space on the ASIC, the REG_CONFIG register is reused to write the fuses. See Table 43. Since the writing of the fuses is a production step, this does not affect the probe during normal operation. To write fuses, a V_{DD5} voltage of 5 V is required, with peak currents of 45 mA for a duration of 10 µs. The following steps need to be taken to correctly write a fuse:

- The FuseWriteEn bit has to be set to ‘1’ in the REG_BIST register (Table 17).
- The REG_CONFIG register needs to be set up with the correct values.
  - The correct fuse value and address need to be configured in the WriteData parameter.
  - The MagicKey parameter needs to be set to 0x42.
  - The FuseWriteCount needs to be set to 125 (10 µs).
- The FuseWriteStart command needs to be sent to write a single fuse. If the MagicKey parameter is not set correctly or the FuseWriteEn bit has not been asserted, this command will be ignored.

When executed correctly, the fuses can be written. It is not possible to undo the writing of a fuse.

Table 43: REG_CONFIG secondary purpose.

Address	# Bits	Name	Description
<7:0>	8	WriteData	Fuse data to write. Fuses can be written (blown) one at a time. Care must be taken such that the written fuse value does not contain more than one extra ‘0’ when compared to the current fuse value. Several cycles might therefore be required to arrive at the desired value.
<9:8>	2	FuseMemoryAddrWR	Fuse write address
<15:10>	6	-	Reserved, must be set to 0
<23:16>	8	FuseWriteCount	Length of the fuse write pulse in steps of 80 ns. (Recommended = 10 µs)
<31:24>	8	MagicKey	Needs to be set to 0x42 to enable the writing of fuses
<55:32>	24	-	Reserved, must be set to 0

8.3.3. System Clock & Data

The US^X-ASIC uses a differential clock/data input compatible to LVDS with reduced common-mode input range. Table 44 contains the system clock specification parameters.

Table 44: System clock specification parameters.

No.	Parameter	Condition	Symbol	MIN	TYP	MAX	UNIT
1	Propagation time		DC_PROPTIME	-	1.8	-	ns
2	LVDS termination		DC_TERM	-	100	-	Ω
3	Common-mode voltage		DC_CM	0.8	1.2	1.6	V
4	Differential swing		DC_DIFFSWING	200	350	500	mV
5	Current consumption		DC_CURRENT	-	160	-	µA
6	Propagation time variation over common mode		DC_PROPTIME	-	-	100	ps

8.3.4. Output Driver Periphery

The output driver is split into two DC-coupled unity-gain stages:

- The output driver in the core; used to buffer the µBeamformer described in section 8.2.6
- The output driver in the periphery; used to drive the cable shown in Figure 44.

The output driver in the periphery can be operated in two distinct modes by configuring the CONFIG_DRV register.

- In feedforward mode (FF) the level-shifted signal goes straight to the class-AB amplifier driving the cable
- In feedback mode (FB) a feedback loop is implemented that increases the bandwidth to 10 MHz

In both cases the signal buffered by the core is level-shifted up to 900 mV to allow a higher output swing. Furthermore, the power consumption is equal for both operating modes.

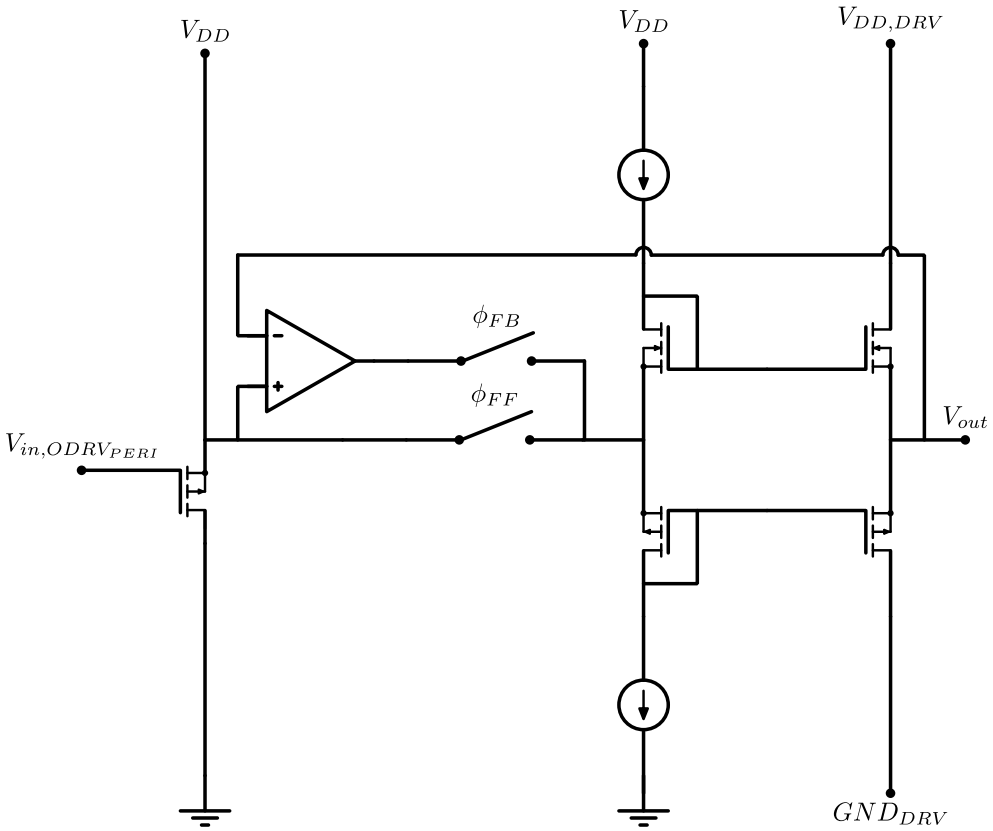


Figure 44: Simplified output driver periphery schematic.

The most important design/performance parameters are summarized in Table 45.

Table 45: $ODRV_{PERI}$ design/performance parameters.

Symbol	Parameter	Condition	Min	Typ	Max	Unit
$ODRV_{PERI,A}$	Gain	@ 5 MHz		0.98		V/V
$ODRV_{PERI,fc}$	Cutoff frequency	C_L = 200 pF		10.8		MHz
$ODRV_{PERI,DC,OP}$	DC operating point at the output	$V_{IN,DC}$ = 526 mV		918		mV
$ODRV_{PERI,Vin,n}$	Noise at the input of the $ODRV_{PERI}$	2.25 - 8.00 MHz		49.30		μV_{RMS}
$ODRV_{PERI,Sn}$	Noise spectral density	2.25 - 8.00 MHz		6.70		$nV/\sqrt{Hz}$
$ODRV_{PERI,NF}$	Noise figure			0.3		dB
$ODRV_{PERI,THD,3M}$	THD w.r.t. a 3 MHz tone	750 mV_{pp}		37		dB
$ODRV_{PERI,DR}$	Dynamic range	Including TGC (20 dB)		94		dB

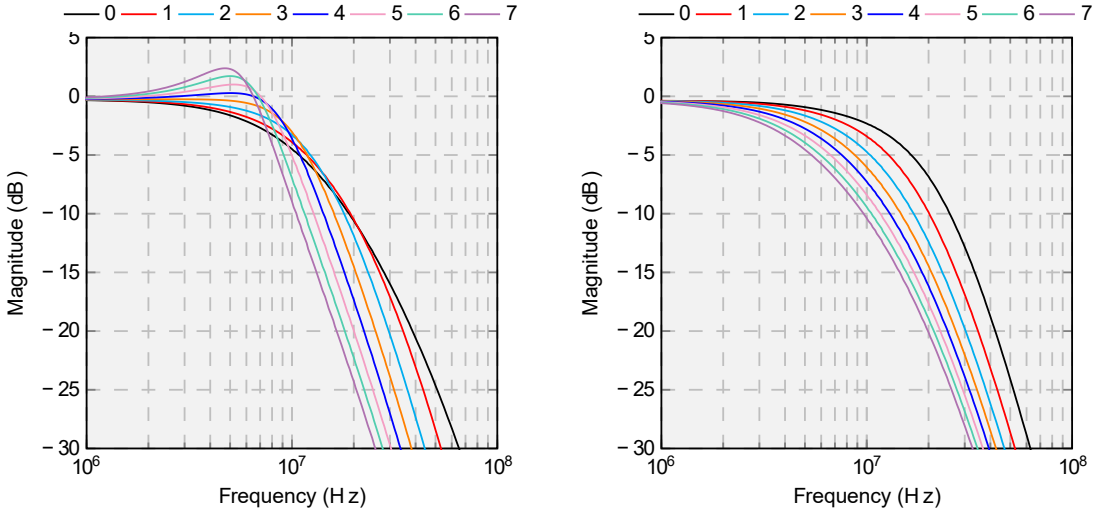


Figure 45: BW w.r.t. current. FB (left) and FF (right).

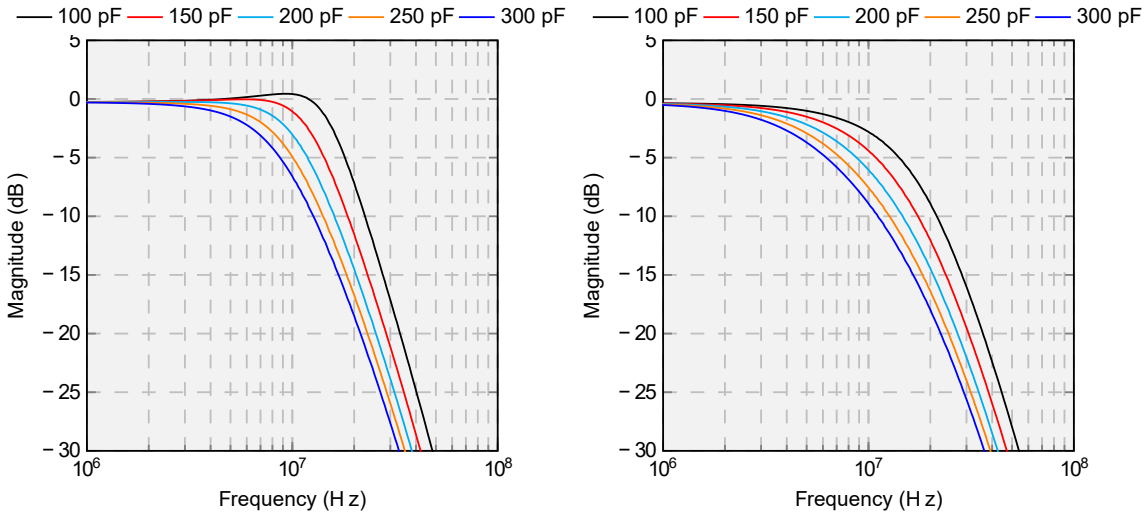


Figure 46: BW w.r.t. C_{LOAD}. FB (left) and FF (right).

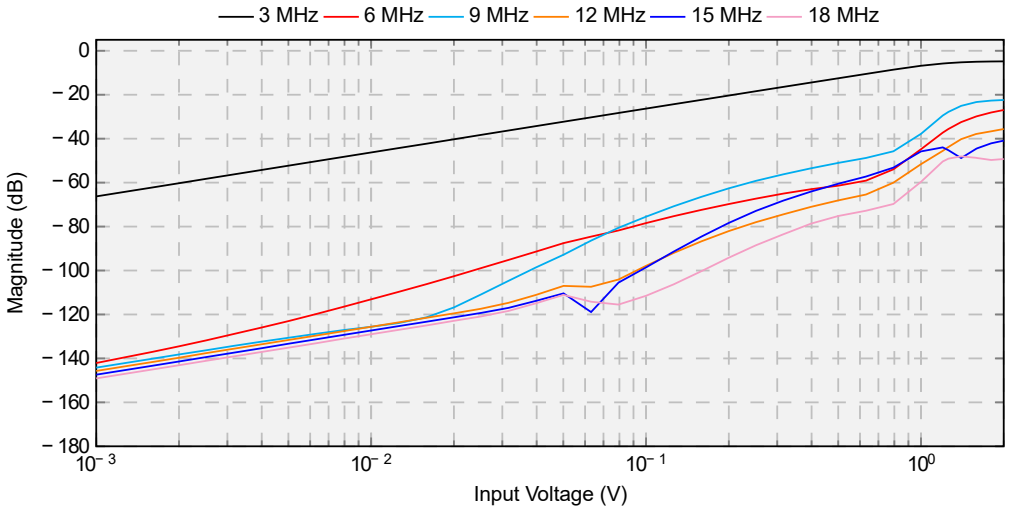


Figure 47: Harmonics w.r.t. a 3MHz input signal in FB mode.

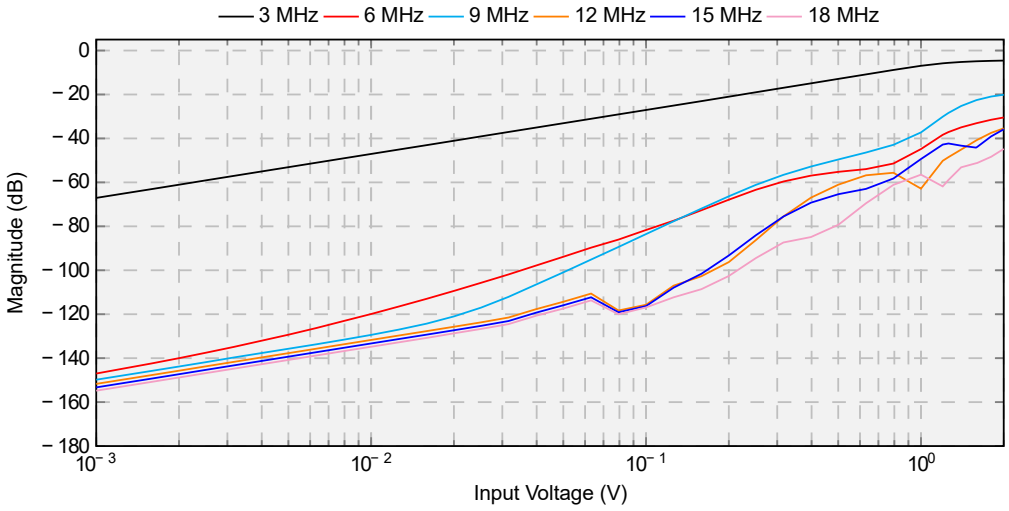


Figure 48: Harmonics w.r.t. a 3MHz input signal in FF mode.

8.3.5. TX Delay Decompression

In order to reduce the setup time of the transmit delays, an approximation of a third order polynomial function is implemented in the ASIC. The ASIC can generate the transmit delays for the complete ASIC from this polynomial, saving roughly 14 µs per ASIC for the writing of the data. The calculation of the delays takes 6.1 µs but can be performed in parallel with the receive decompression. Furthermore, the decompression operations can be performed concurrently for all ASICs. The calculation of the delays is divided into the calculation of group center delays and the calculation of individual element delays within a group. The group center delays are generated by the following polynomial function:

$$d_{group}[X,Y] = (c_0 - 0.5) + c_1 L_1[X] + c_2 L_2[X] + c_3 L_3[X] + c_4 L_1[Y] + c_5 L_2[Y] + c_6 L_3[Y] + c_7 L_4[X] L_4[Y]$$

Where X and Y are group coordinates of the ASIC(s) with $X \in \{0,1,...,15\}$ and $Y \in \{0,1,...,15\}$, c_n are user defined constants and L_N represent 4 curves implemented as lookup tables. Every ASIC has 16 groups in the X direction and 4 groups in the Y direction. Since up to 4 ASICs can be combined into a larger array, the total number of groups in the Y direction is also 16. The center of the array is independent of the number of ASICs in the array and is always defined at X,Y = 7.5,7.5. The shift is implemented by the 'TXDELAY_SHIFT' parameter in the REG_CONFIG register (Table 22). E.g. for one ASIC $Y \in \{6,7,8,9\}$ and for 2 ASICs $Y \in \{4,5,...,11\}$. The element delays are given by:

$$d_{el}[X,Y,x_{el},y_{el}] = dX[X,Y] \cdot (x_{el} - 1.5) + dY[X,Y] \cdot (y_{el} - 1.5) + 16$$

Where x_{el} and y_{el} are the coordinates of elements within a group with both x_{el} and $y_{el} \in \{0,1,2,3\}$ and dX and dY are given by:

$$dX(X,Y) = (2c_1 + c_2L_1[X] + c_3L_2[X] + c_7L_1[Y])/32$$
$$dY(X,Y) = (2c_4 + c_5L_1[Y] + c_6L_2[Y] + c_7L_1[X])/32$$

Table 46: Transmit lookup table.

N	L_1*64	L_2*64	L_3*64	L_4*8√2
0	-120	56	-18	-15
1	-104	42	-12	-13
2	-88	30	-7	-11
3	-72	20	-4	-9
4	-56	12	-2	-7
5	-40	6	-1	-5
6	-24	2	0	-3
7	-8	0	0	-1
8	8	0	0	1
9	24	2	0	3
10	40	6	1	5
11	56	12	2	7
12	72	20	4	9
13	88	30	7	11
14	104	42	12	13
15	120	56	18	15

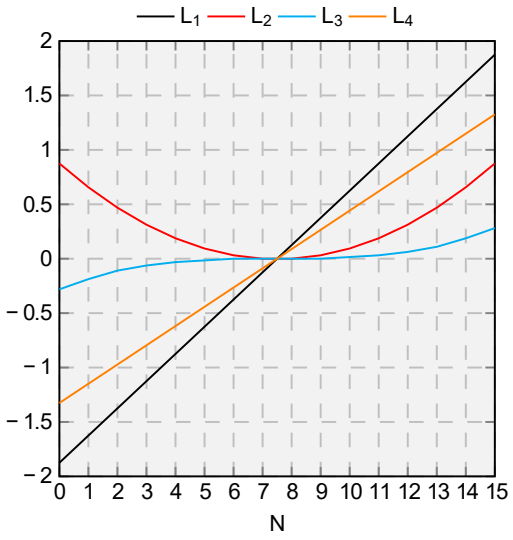


Figure 49: Lookup table plots.

The coefficients $c_{0..7}$ are stored in REG_TXDELAY1 and REG_TXDELAY0 (Table 18). An example is shown in Figure 50 and Figure 51 with the original and decompressed delays for one ASIC with a focus at 50 mm without steering. The delays were generated with the following coefficients:

$$c_0 = 17 \quad c_1 = 0 \quad c_2 = -18 \quad c_3 = 0 \quad c_4 = 0 \quad c_5 = -18 \quad c_6 = 0 \quad c_7 = 0$$

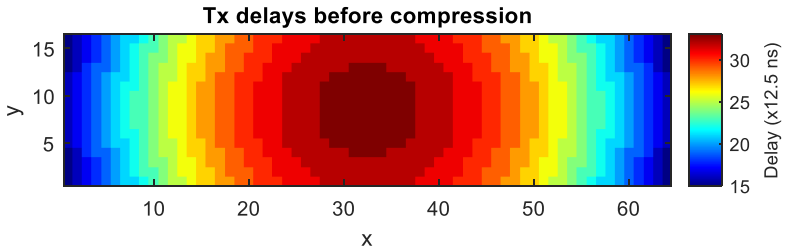


Figure 50: Original TX delays where x and y represent the element numbers.

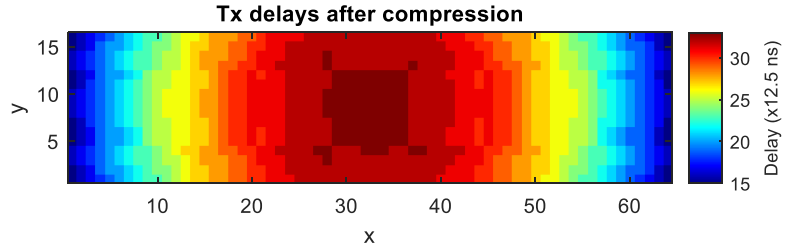


Figure 51:Compressed polynomial TX delays where x and y represent the element numbers.

The MATLAB script used to generate this example is attached in Appendix B.1.

8.3.6. RX Delay Decompression

In order to reduce the setup time of the receive delays, an approximation of a second order polynomial function is implemented in the ASIC. The ASIC can generate the receive delays for the complete ASIC from this polynomial, saving roughly 20 µs per ASIC for the writing of the data. The calculation of the receive delays takes 5.5 µs but can be performed in parallel with the transmit decompression. Furthermore, the decompression operations can be performed concurrently for all ASICs. Unlike transmit however, only element delays are configured on the ASIC. The center (group) delays are applied on the ultrasound system itself. The element delays are defined by:

$$d_{el}[X,Y,x_{el},y_{el}] = \left(dX[X]\cdot(x_{el}-1.5)+dY[X,Y]\cdot(y_{el}-1.5)\right)\cdot\frac{c_9}{4}+\frac{c_7}{16}\cdot(x_{el}-1.5)+\frac{c_8}{16}\cdot(y_{el}-1.5)+14$$

Where dX and dY are given by:

$$dX[X] = \frac{c_0}{32}+c_1L_1[X]+c_2L_2[X]$$
$$dY[X,Y] = \frac{c_3}{32}+c_4L_1[Y]+c_5L_2[Y]+c_6L_1[X]$$

Where L_N represents first and second order discrete curves.

Table 47: Receive lookup table.

N	L_1*1024	L_2*1024
0	-30	28
1	-26	21
2	-22	15
3	-18	10
4	-14	6
5	-10	3
6	-6	1
7	-2	0
8	2	0
9	6	1
10	10	3
11	14	6
12	18	10
13	22	15
14	26	21
15	30	28

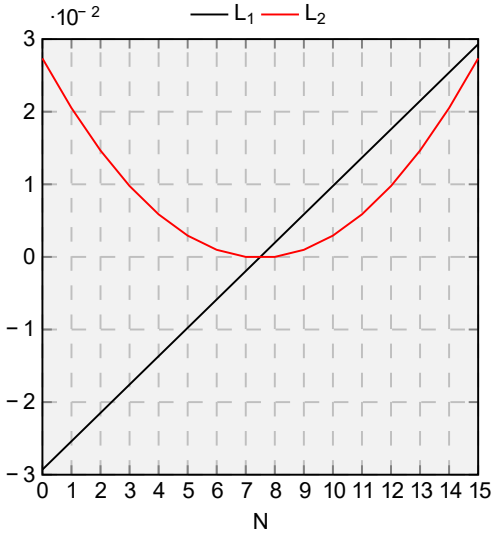


Figure 52: Lookup table curves.

The coefficients $c_{0..9}$ are stored in REG_RXDELAY1 and REG_RXDELAY0 (Table 19). An example is shown in Figure 53 with the original and decompressed delays for one ASIC with a focus at 10 mm with steering (α_x,α_y) = (30°,0°). The delays were generated with the following coefficients:

$$c_0=0 \quad c_1=-42 \quad c_2=-18 \quad c_3=0 \quad c_4=-56 \quad c_5=0 \quad c_6=0 \quad c_7=48 \quad c_8=0 \quad c_9=8$$

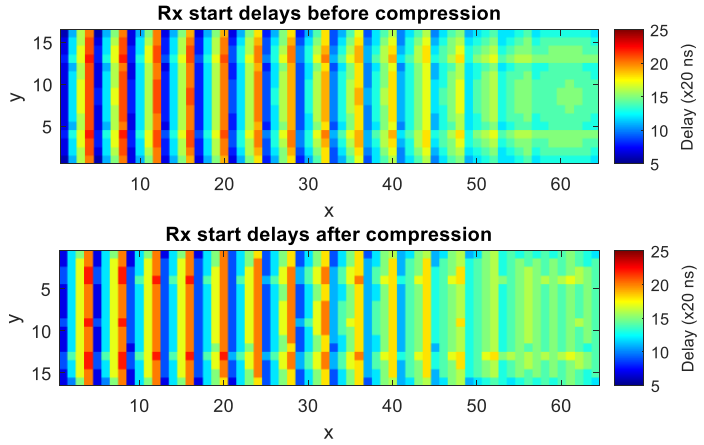


Figure 53: Original and compressed polynomial RX delays where x and y represent the element numbers.

8.3.7. Dynamic Receive

The dynamic receive configuration uses a master curve M(t) that is an 8-point piecewise linear function over time and 8 steering curves $C_{0...7}(t)$ that are multiples (fractions) of the master curve. The steering curves are connected to the elements and control their dynamic receive behavior. The easiest way to describe both the piecewise-linear master curve and the 8 steering curves is with the following pseudo code:

```
t_elapsed=0;
for (int segment=0; segment<8; segment++){
  if (t>=t_elapsed+Duration[segment]){
    M+=Duration[segment]*Slope[segment];
  }
  else{
    M+=(t-t_elapsed)*Slope[segment];
    break;
  }
  t_elapsed+=Duration[segment];
}
for (int i=0; i<8; i++){
  M[i]=(MultiSign[i]*Multiplier[i]*M)/256 + StartPhase[i];
}
```

The parameters used in the curve generation are configurable in REG_RXDYN (Table 20). The assignment of the elements to 1 of the 8 curves is described by:

$$A[X,Y,x_{el},y_{el}] = dX[X]\cdot(x_{el}-1.5)+dY[X,Y]\cdot(y_{el}-1.5)+3.5$$

Where the result is rounded to the nearest integer. An example of the curve assignment is shown in Figure 54 for one ASIC with a focus (start delays) at 10 mm with steering (α_x,α_y) = (30°,0°). The delays were generated with the following coefficients:

$$c_0=0 \quad c_1=-42 \quad c_2=-18 \quad c_3=0 \quad c_4=-56 \quad c_5=0 \quad c_6=0 \quad c_7=48 \quad c_8=0 \quad c_9=8$$

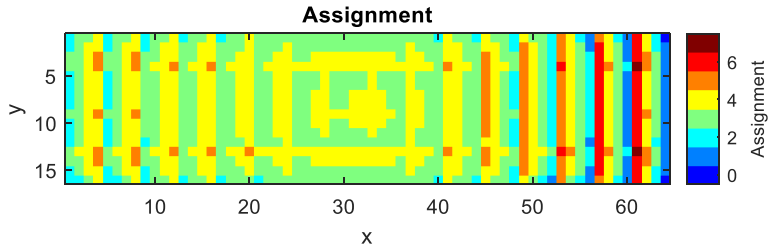


Figure 54: Example of curve assignment for 50 mm focus and a steering angle (α_x,α_y) = (30°,0°).

An example of the dynamic delays over time is shown in Figure 55 using the constants defined in Table 48 and Table 49 with RunRxTime set to 29. These dynamic delays are added on top of the element delays described in the previous section.

Table 48. Settings for dynamic curves

Curve number	Multiplier	MultiSign*	StartPhase
0	7	1	4
1	5	1	4
2	3	1	4
3	1	1	4
4	1	0	4
5	3	0	4
6	5	0	4
7	7	0	4

*1=increase; 0=decrease

Table 49. Segments of the master curve

Segment	Duration	Slope
0	4	16
1	1	169
2	3	91
3	5	38
4	7	16
5	8	7
6	11	4
7	12	2

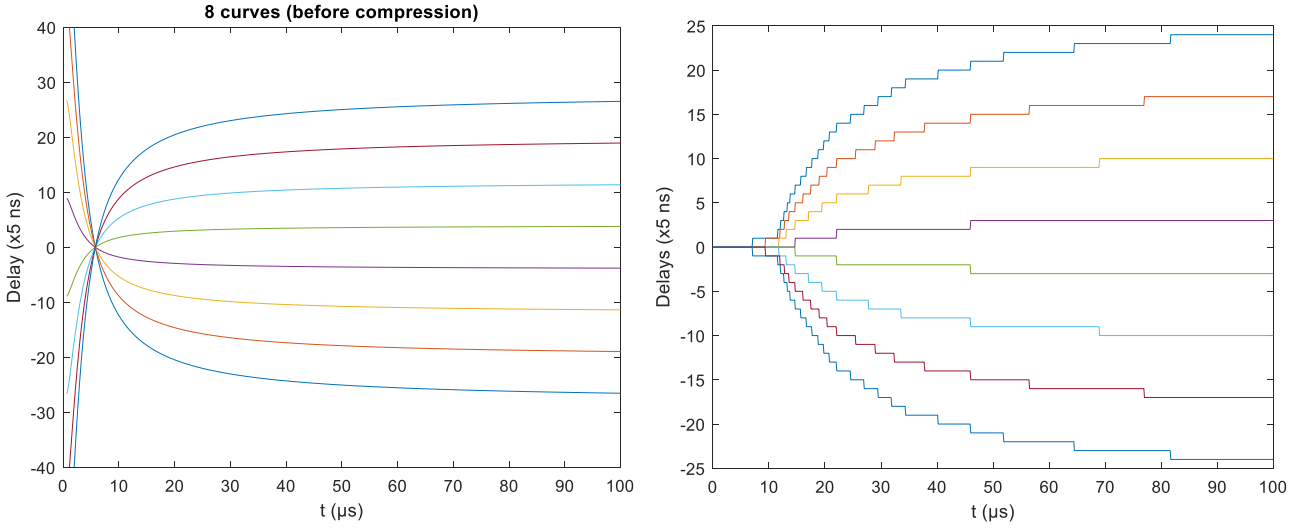


Figure 55: Original and compressed dynamic RX delays.

8.3.8. Built-In Self-Test

For the purpose of basic production testing, a BIST module is integrated into the ASIC. The module is controlled by the REG_BIST register described in section 7.1.2 and is not intended for ultrasound purposes. The core of the BIST is a Time-to-Digital Converter (TDC) implemented as a Single-Slope ADC connected to a 16-bit counter. The counter runs in the periphery and is clocked on both rising- and falling edges, achieving a resolution of 5 ns.
The ADC can be connected to different parts of the ASIC in order to measure:

- Internal operating points
- Internal capacitance
- External capacitance
- Internal temperature sensor

Connection to the desired nodes is achieved through the global continuous-wave lines: `CW_CH_RX<3:0>` and `CW_CH_TX<3:0>`. The BIST ADC can be connected to the LNA inputs, LNA outputs, output driver inputs and output driver outputs as shown schematically in Figure 56.

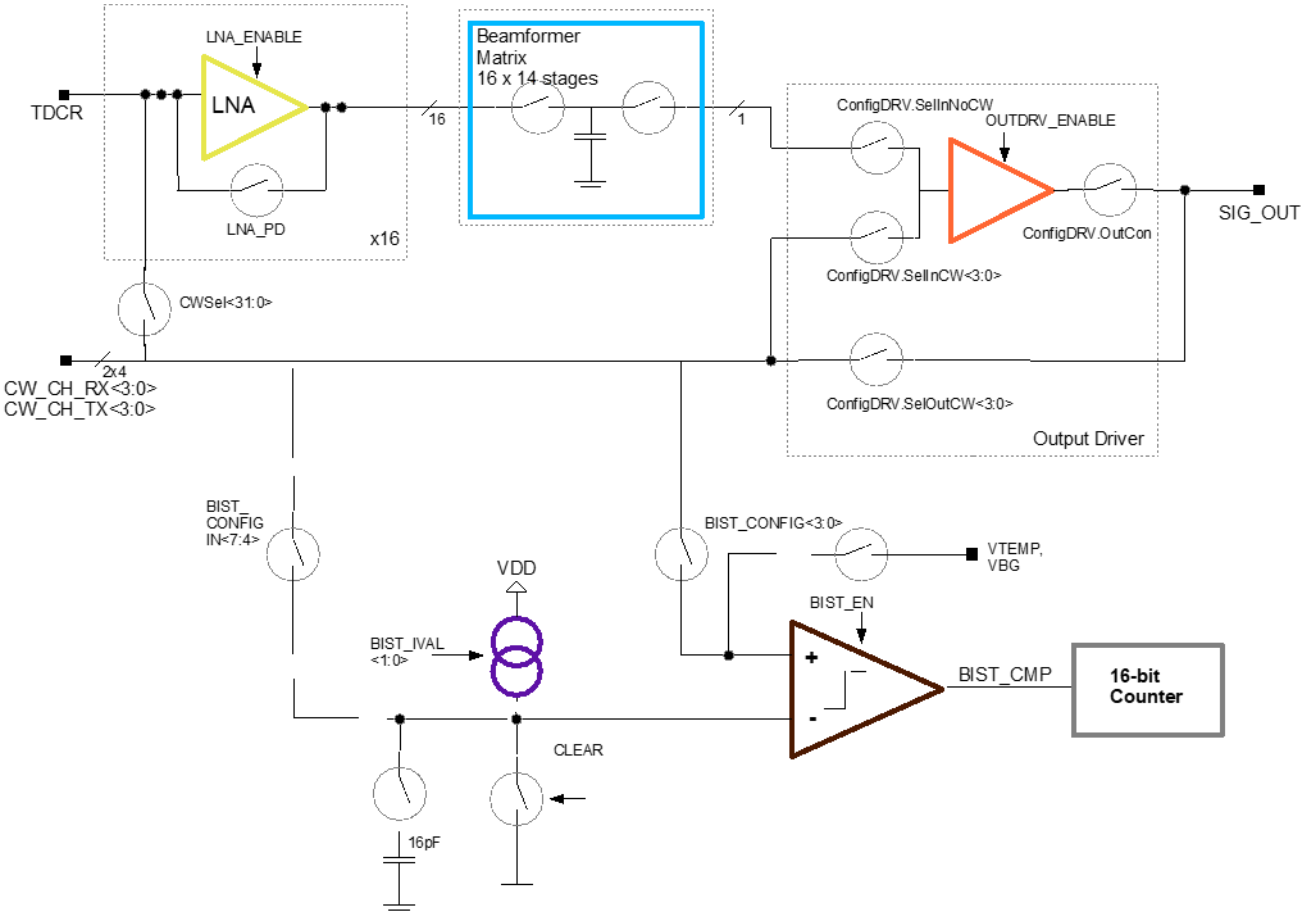


Figure 56: Schematic overview of the available BIST connections.

9. Device Operation

9.1. Initialization

To ensure correct operation of the US^X-ASIC the following steps need to be taken:

- Write DataOutClkDiv in the REG_BIST register to set up the intended readback frequency.
- Read the QSPI_NERROR pin to ensure that correct communication has been established.
- Read the BIST error status by sending the BistGetErrorStatus command.
- Read out the 32-bit fuses to get:
  - The serial number of the US^X-ASIC (not required).
  - The current calibration value.
- Write the current calibration value into the IBIAS_CAL parameter in the REG_BIST register.

9.2. Global Setup

After the initialization has completed successfully, it is up to the user to configure the rest of the ASIC. Apart from all the registers that exist for specific operating mode setup, there are two registers that are used to setup various (shared) parameters like operating points, currents, enable signals and other parameters that control the behavior of the ASIC:

- REG_CONFIG (Table 22)
- CONFIG_DFF (Table 30)

Table 50 provides a summary of the most important parameters stored in these registers. They can be used as starting points as far as bandwidth, noise, and power consumption go. Unless stated otherwise, these settings have also been used to simulate/measure the analog performance of the different modules (Chapter 8). It is recommended to adapt these parameters to obtain an optimum performance for each operating mode.

Table 50: Recommended starting values.

Register	Name	Value	Notes
REG_CONFIG	(Table 22)		
	TxClkDivider	20	40 MHz → 25 ns TX_{CLK} period
CONFIG_DFF	(Table 30)		
	ISEL_LNA	4	30 µA
	ISEL_ODRV	3	40 µA
	ISEL_RESCNTL	6	Insignificant
	ISEL_DCGND	8	160 µA
	ISEL_PCHVP	8	160 µA
	RES_CAL	4	600 mV
CONFIG_DRV	(Table 32)		
	DRV_BIASSEL	4	260 µA
	DRV_FF_nFB	0	Feedback mode

9.3. Transmit Setup

Setting up the ASIC for waveform transmission can be divided into several steps:

- Transmit clock generation
- Transmit waveform generation
- Transmit focusing

All steps will be discussed in the following subsections.

9.3.1. Transmit Clock Generation

All transmit delays and waveform generation utilize a TX_{CLK}. This TX_{CLK} is a fractional division of the 100 MHz ASIC clock (CLK_{ASIC}) and its frequency is defined by:

$$F_{TX_{CLK}} = \frac{100\ MHz}{TxClkDivider}$$

Where TxClkDivider is in fixed point data format with 5 integer and 3 fractional bits. The division is done digitally and is synchronized to both rising and falling edges of the CLK_{ASIC}, resulting in a delay resolution of 0.5 TX_{CLK}. No internal oscillator or PLL is used. When defining the transmit clock, the following parameters should be taken into account:

- Maximum delay: The TX_{CLK} period should be chosen such that the complete group can be covered for the maximum steering angle. The maximum delay is defined by:

  $$T_{DELAY,MAX} = 29 \cdot \frac{TX_{CLK}}{2}$$

  Where 29 represents the maximum programmable value for the transmit delays (section 7.2.2). For a ±30° scan in either X or Y direction (focus within a group aimed at infinity), the delay required to cover the complete group is:

  $$T_{DELAY,30°} = \frac{3 \cdot 180\ \mu m \cdot \sin(30°)}{1540\ {}^{m}/_{s}} = 175\ ns$$

- Waveform resolution: The TX_{CLK} period should be chosen fine enough such that there is enough resolution to generate the desired waveforms. A tradeoff exists between resolution and maximum steering angle.
- For focusing close by (less than 50 mm), the group offset delays must also be covered for the maximum steering angle:

  $$T_{DELAY,MAX} = 255 \cdot TX_{CLK}$$

- Jitter: No delay should be programmed that results in jitter due to division.

Some examples that indicate various usages of the TxClkDivider parameter:

- 16 = 0x10 = 0b00010000 = 50 MHz (all 20 ns periods, uses only rising edges of the clock)
- 20 = 0x14 = 0b00010100 = 40 MHz (all 25 ns periods, uses also falling edges of the clock)
- 23 = 0x17 = 0b00010111 = 34.7826 MHz (1x25 ns period 3x30 ns periods)
- 24 = 0x18 = 0b00011000 = 33.3333 MHz (30 ns periods, uses only rising edges of the clock)

Table 51 contains a range of TxClkDivider values that result in transmit waveforms with center frequencies between 2.5 – 6.25 MHz that are constructed with 4 – 8 samples per period.

Table 51: TxClkDivider values for a center frequency between 2.5 – 6.25 MHz.

Input Frequency 1,00E+008 (100 Mhz)

Samples per Periode	4	5	6	7	8
Resolution (deg)	90	72	60	51,43	45

TxClkDivider	F_TX(MHz)	F_XMIT(MHz)	F_XMIT(MHz)	F_XMIT(MHz)	F_XMIT(MHz)	F_XMIT(MHz)	1/F_TX(ns)	max. TXDelay (ns)
40	20,00	5,00	4,00	3,33	2,86	2,50	50,00	725
39	20,51	5,13	4,10	3,42	2,93	2,56	48,75	707
38	21,05	5,26	4,21	3,51	3,01	2,63	47,50	689
37	21,62	5,41	4,32	3,60	3,09	2,70	46,25	671
36	22,22	5,56	4,44	3,70	3,17	2,78	45,00	653
35	22,86	5,71	4,57	3,81	3,27	2,86	43,75	634
34	23,53	5,88	4,71	3,92	3,36	2,94	42,50	616
33	24,24	6,06	4,85	4,04	3,46	3,03	41,25	598
32	25,00		5,00	4,17	3,57	3,13	40,00	580
31	25,81		5,16	4,30	3,69	3,23	38,75	562
30	26,67		5,33	4,44	3,81	3,33	37,50	544
29	27,59		5,52	4,60	3,94	3,45	36,25	526
28	28,57		5,71	4,76	4,08	3,57	35,00	508
27	29,63		5,93	4,94	4,23	3,70	33,75	489
26	30,77		6,15	5,13	4,40	3,85	32,50	471
25	32,00			5,33	4,57	4,00	31,25	453
24	33,33			5,56	4,76	4,17	30,00	435
23	34,78			5,80	4,97	4,35	28,75	417
22	36,36			6,06	5,19	4,55	27,50	399
21	38,10				5,44	4,76	26,25	381
20	40,00				5,71	5,00	25,00	363
19	42,11				6,02	5,26	23,75	344
18	44,44					5,56	22,50	326
17	47,06					5,88	21,25	308
16	50,00					6,25	20,00	290

9.3.2. Transmit Waveform Generation

Once the TX_{CLK} has been configured, the transmit waveforms can be specified. The XMITWF_DFF register (section 7.2.6) contains waveform configuration parameters that are used to configure two transmit waveforms per group. To increase the versatility of the waveforms, the waveforms are split into three distinct parts:

- Head: Can be used to define a 'startup' sequence for the waveforms
- Body: Contains the main waveform that is to be transmitted (and repeated)
- Tail: Can be used to define an ending sequence that should follow the body.

The contents of the XMITWF_DFF register are passed to the waveform generator. The internal process of generating the transmit waveforms from the configured parameters has been described in detail in section 8.2.2. An example is shown in Figure 57 to further clarify the process.

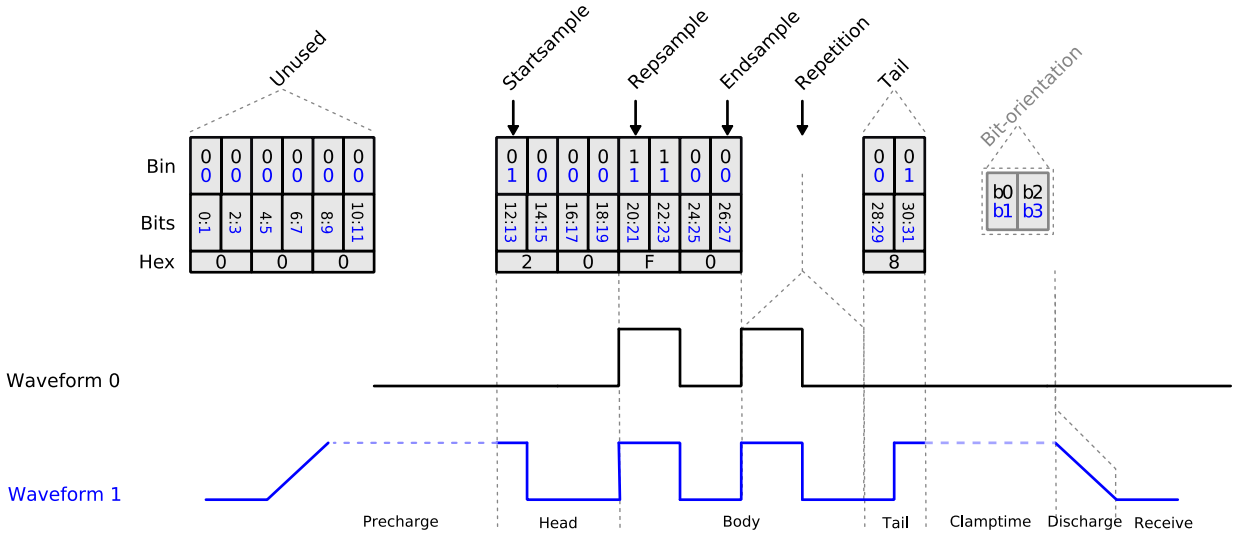


Figure 57: Waveform generation example.

Waveform 0 is a simple waveform with 2 pulses, whereas waveform 1 is slightly more complicated because it starts and ends in a 'high' state. Note that the waveforms have been selected for the purpose of explanation, not for their acoustic properties. The parameters extracted from Figure 57 are shown in Table 52.

Table 52: Waveform generation example parameters.

Name	Value	Explanation
TX_STARTSAMPLE	6	The head of the waveform that is transmitted before the main pulse is defined at the 7th bit inside both waveforms. Since we start counting at 0, the value is 6. The first 6 bits inside each waveform positions (0:11) are not used in this example but can be used freely. Waveform 0 starts in a 'low' state, whereas waveform 1 starts in a 'high' state. In this particular example, waveform 1 is also pre-charged. The user can change this behavior be setting TX_PRECHARGE to the desired option.
TX_REPSAMPLE	10	The main pulses are defined at bits 10:13.
TX_ENDSAMPLE	13	
TX_PATTERNREP	1	The main pulses are repeated once after the initial transmission.
TX_WF_SETUP	0x80F02000	The binary string belonging to the waveform displayed in the figure is specified from LSB to MSB. When this binary value is converted to a hexadecimal string we obtain the value 0x80F02000.

It should also be noted that the depicted waveforms can be configured in multiple ways. For example by including the repetition into the main waveform and reducing the number of repetitions to 0. The other parameters would need to be adjusted accordingly.

9.3.3. Transmit Focusing

Transmit focusing is achieved by configuring the transmit delays in the TXDELAY_DFF register (section 7.2.2). There are two ways to configure the transmit delays:

- Directly writing the TXDELAY_DFF registers that contain the group center delays and element delays
- Specifying a polynomial in the REG_TXDELAY1 and REG_TXDELAY0 registers (section 7.1.3) that is decompressed into delays that are written into the TXDELAY_DFF registers.

The first option has the advantage that every single delay can be written manually and the highest possible accuracy can be achieved. The disadvantage is that directly writing all delays takes 14.16 µs, during which transmit and receive operations are not possible. The second option has the advantage that writing the delays takes only 0.24 µs. The decompression requires only 5.5 µs and can be run in parallel with the RX decompression and can also run concurrently on multiple ASICs. A disadvantage is that delays are limited to the accuracy of the decompression implemented in the ASIC. The decompression has been described in more detail in section 8.3.5.

9.4. Receive Setup

Setting up the ASIC for receive can be divided into several steps:

- Receive clock generation
- Receive focusing (and dynamic receive focusing)
- Output-driver setup

All steps will be discussed in the following subsections.

9.4.1. Receive Clock Generation

The µBeamformer runs on the 25 MHz RX_{CLK}, which is derived from the 100 MHz CLK_{ASIC}.

9.4.2. Receive Focusing

Receive focusing is achieved by configuring the receive delays in the RXDELAY_DFF register (section 7.2.3). There are two ways to configure the receive delays:

- Directly writing the RXDELAY_DFF registers that contain the group center delays and element delays
- Specifying a polynomial in the REG_RXDELAY1 and REG_RXDELAY0 registers (section 7.1.4) that is decompressed into delays that are written into the RXDELAY_DFF registers.

The first option has the advantage that every single delay can be written manually and the highest possible accuracy can be achieved. The disadvantage is that directly writing all delays takes 20.56 µs, during which transmit and receive operations are not possible. The second option has the advantage that writing the delays takes only 0.40 µs. The decompression requires only 6.1 µs and can be run in parallel with the TX decompression and can also run concurrently on multiple ASICs. A disadvantage is that delays are limited to the accuracy of the decompression implemented in the ASIC. The decompression has been described in more detail in section 8.3.6.

9.4.2.1. Dynamic Receive Focusing

Dynamic receive focusing has been explained in section 8.2.5.2. and can be used to alter the steering angle over time. This could lead to an improved image quality in the nearfield.

9.4.3. Output-Driver Setup

Every output driver periphery has a configuration register (CONFIG_DRV) of 16 bits. There are 64 instances of this register in total and is used to enable the output drivers and configure them for different operating modes. Figure 58 gives a schematic overview of the available switches that can be used to set up the output driver periphery for different operating modes.

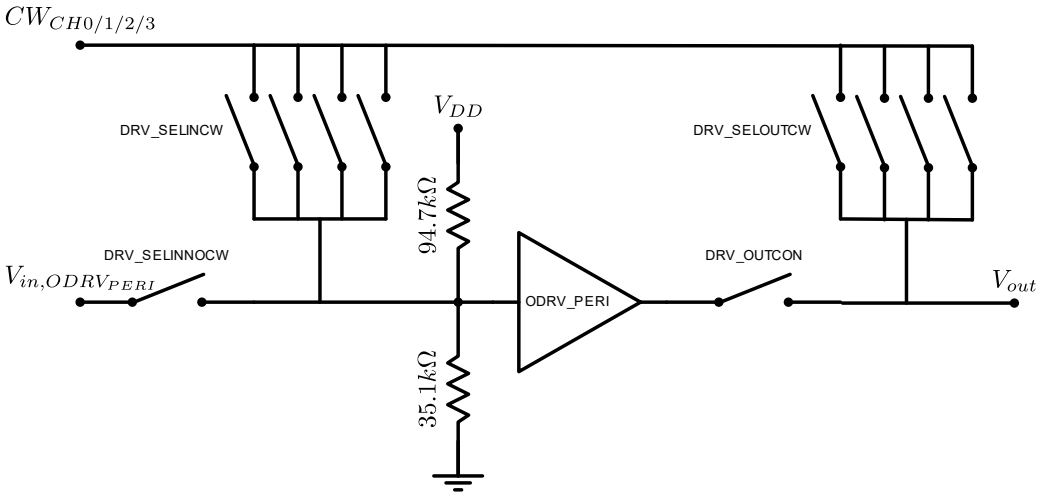


Figure 58: Output driver periphery CW-selection scheme.

Table 53 gives an overview of multiple output driver configurations for different operating modes. It should be noted that the DRV_SELINCW and DRV_SELOUTCW parameters must be one-hot encoded. If not, two or more CW channels will be shorted. For CW transmit, the DC-operating point will be set by the ESD diode to GND_DRV at the begin of a CW transmission. Therefore, if possible the CW transmit voltage should be ramped up.

Table 53: CONFIG_DRV configurations for different operating modes.

Operating Mode	Description	DRV_ENABLE	DRV_BIAS_SEL	DRV_SELINCW	DRV_SELOUTCW	DRV_OUTCON	DRV_SELINNOCW	DRV_RES_EN
Default	Default (low-power) case at power-up. Driver is disabled and tri-stated.	0	000	0000	0000	0	0	0
Normal	Signal cable is connected to the driver output.	1	011	0000	0000	1	1	0
CW Transmit	Output driver periphery is inactive. V_{out} is directly connected to 1 of 4 CW lines.	0	000	0000	0001/ 0010/ 0100/ 1000	0	0	0
CW Receive Passive	Output driver periphery is inactive. V_{out} is directly connected to 1 of 4 CW lines.	0	000	0000	0001/ 0010/ 0100/ 1000	0	0	0
CW Receive Active	Output driver periphery is active and drives the signal cable.	1	011	0001/ 0010/ 0100/ 1000	0000	1	0	1
Cap. measurement	Measurement of output capacitance.	0	000	0000	0001	0	0	0
Op.point measurement	Measurement of the output operating point.	1	011	0000	0001	1	0	1

9.5. Pulsed Mode

To operate the US^X-ASIC in pulsed mode, it is important to understand the internal timing characteristics. All timing critical operations are internally controlled by a Finite State Machine (FSM) that cycles through 3 different states: Idle, Setup and Run.

In the Idle state, all configuration parameters including the transmit and receive polynomials can be set up as described in the previous sections. After the ASIC has been configured, a beam (delay configuration) is loaded with a TriggerPageXX command (denoted as CmdP in the figure below). If the BeamRunTxCalc and BeamRunRxCalc parameters have been set in the REG_CONFIG register, the decompression of the polynomials is started, and the FSM moves to the Setup state. It is also possible to directly write the RXDELAY_DFF and TXDELAY_DFF registers, and to disable BeamRunTxCalc and BeamRunRxCalc. This could be useful in modes where the delays are unchanged.

In the Setup state the FSM waits for the decompression to complete. The transmit and receive decompression run in parallel and take 6.1 µs and 5.5 µs respectively during which the delays are written into the TXDELAY_DFF and RXDELAY_DFF registers. When the decompression is complete, a TriggerRun command (CmdT) is required to load the beam, which is on hold.

In the Run state, a timer is running that controls 4 timing critical activities that are configured by the user:

- Setup of the µBeamformer
  - Delays are transferred from the RXDELAY_DFF to the MATRIX_DFF registers that control the µBeamformer.
  - Starts at SetupRxTime and takes 1.2 µs.
  - Can also be started after a TriggerPageXX command by setting the ExtRxSetupTime to ‘1’. SetupRxTime must then be set between 5.5 µs and 6.1 µs for this to have any effect. This can save up to 560 ns.
- Transmit clock generator
  - Starts at RunTxTime and stops at StopTxTime.
  - Can also be started after a TriggerPageXX command by setting ExtTxRunTime to ‘1’. The transmit clock must be kept active long enough, to transmit with all elements. Depending on the delays, a different time might be required. If elements need to be precharged, that needs to be taken into account as well.
- TGC
  - Starts at TGCStart.
  - The time between gain steps is defined by TGCIncrement.
- Receive clock generator
  - Starts at RunRxTime and stops at StopRxTime.
  - It is recommended (but not required) to start the receive clock generator after the setup of the µBeamformer has completed. For debugging purposes, it is possible to keep the receive clock active beyond the StopRxTime by setting the BeamContRx to ‘1’.
  - After the receive clock has stopped, all other activities are also stopped and the FSM returns to the Idle state.

A timing diagram is shown in Figure 59 for clarification.

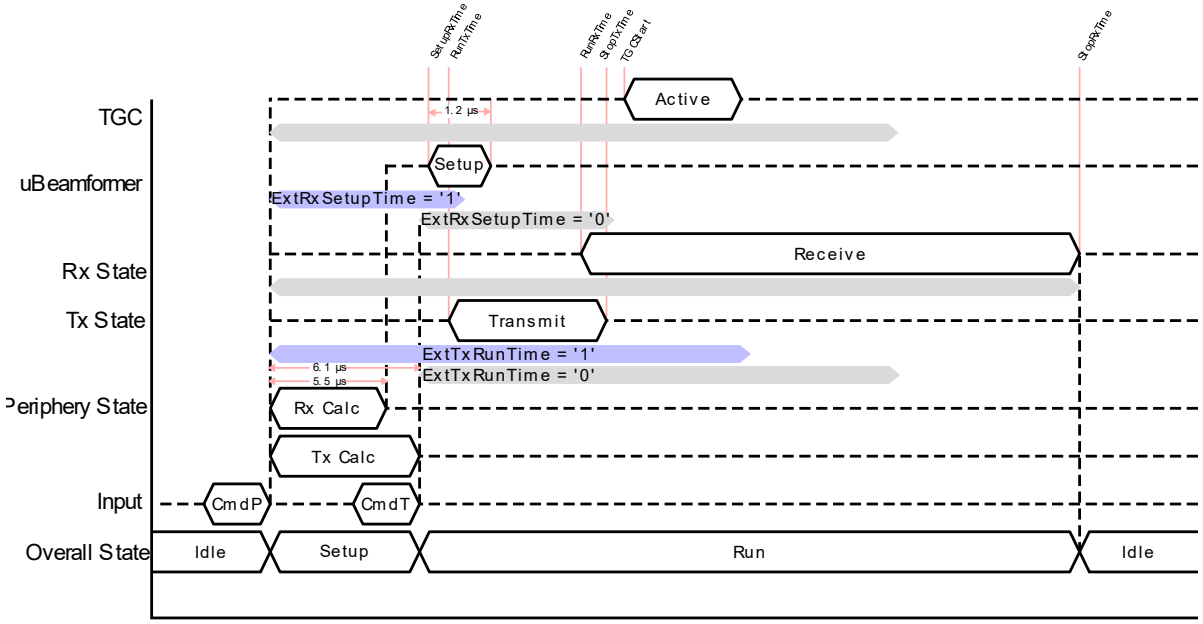


Figure 59: Pulsed mode timing diagram.

9.6. Pulsed Doppler Mode

The main difference between Pulsed Mode and Pulsed Doppler Mode is that repeatability is more important. Some techniques implemented in the US^{X}-ASIC using a pseudo-random generator to reduce digital artefacts need to be switched off in order to increase the repeatability of beams. Table 54 shows the registers that must be taken into account for Pulsed Doppler Mode.

Table 54: Randomization parameters that could affect Pulsed Doppler Mode.

Register	Pulsed Doppler Setting	Effect on Bmode Images
REG_CONFIG.RandomizeDynUpdate	‘0’	Reduces rings caused by dynamic receive.
REG_CONFIG.RXDELAY_LSBRAND	“00”	Reduces sidelobes in receive beamforming due to calculation quantization.
REG_CONFIG.TXDELAY_LSBRAND	“00”	Reduces sidelobes in transmitbeamforming due to calculation quantization.
REG_RXDYN.RandomOpt	“00”	Reduces rings caused by dynamic receive.
XMITWF_DFF.TX_PRECHARGE	not “01”	Reduces spurious transmit due to element precharge for waveforms starting with a falling edge.
CONFIG_DFF.ISEL_RESCNTL	> 10	Reduces TGC rings in image.
CONFIG_DFF.RANDOM_REG_UPD	‘0’	Reduces µBeamformer pattern noise.
CONFIG_DFF.MATRIX_OFFSET_VAL	0—13	Reduces µBeamformer pattern noise.

9.7. Continuous-Wave Mode

Although Continuous-Wave (CW) operation is technically possible by infinitely transmitting with half the elements and receiving with the other half, the noise, dynamic range and power requirements will not be met. Therefore, a dedicated CW mode is implemented in the US^{X}-ASIC. In this mode, the ASIC is split in the middle where the left half acts as a transmitter and the right half acts as a receiver. It is also possible to use the left half of the ASIC for receive and the right half for transmit. The internal connections are the same, the TX/RX is just the chosen naming convention. This is graphically shown in Figure 60. Furthermore, all electronic circuitry on the ASIC is powered down. Only numerous switches do the CW-beamforming by combining the elements in a fixed beamforming profile with phases of 90° while taking advantage of the arbitrary 360° phase wrapping. In CW mode, there is no inherent relation between the elements contained in a group and that group's acoustic line. On the contrary, in this mode the routing configuration can be viewed as 4 CW lines to which any acoustic channel and any element can be connected to arbitrarily.

Transmission is performed by the system by sending signals over the acoustic lines to the left half of the ASIC. For these groups, the output driver has to be set up in CW transmit mode as shown in Table 53. In this setup the acoustic lines are routed directly to the 512 transducers over the 4 available CW transmit lines (CW_CH_TX<3:0>) that can withstand a maximum voltage of 10 V_{PP}.

Receiving offers the same configurability as transmit. Any element can be connected to either of 4 CW receive lines (CW_CH_RX<3:0>). Furthermore, the output driver can be used to actively drive the signal cables, reducing the load on the receiving elements and therefore increasing the voltage swing and improving the SNR. See Table 53.

In a typical CW operations this translates to:

- Transmit
  - 32 Acoustic channels
  - 8 Acoustic channels for each of the 4 CW transmit lines
    - 64 Elements per CW line
    - 16 Elements per acoustic channel
      - Impedance per acoustic channel ≈ 600 Ω (10 kΩ per XDCR)
- Receive
  - 32 Acoustic channels
  - 8 Acoustic channels for each of the 4 CW receive lines
    - 64 Elements per CW line
    - 16 Elements per acoustic channel

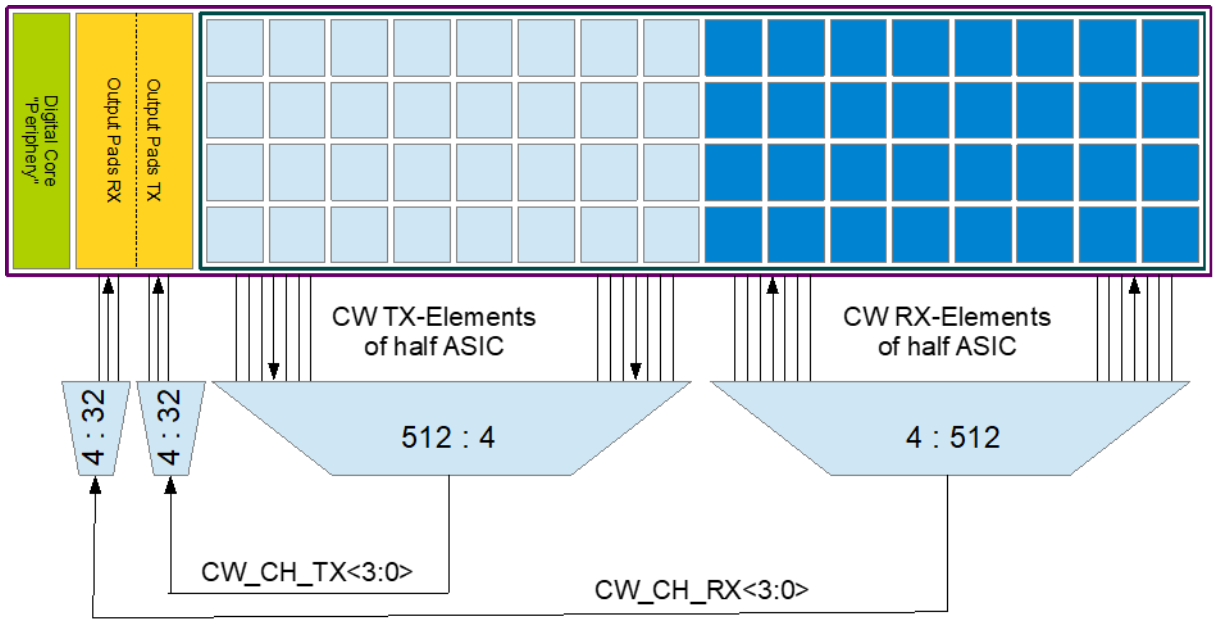


Figure 60: Continuous-Wave mode configuration.

9.7.1. Setup

The following steps need to be taken to set up the US^{X}-ASIC for CW mode:

- CWSEL_DFF:
  - Configure the elements with the desired channel assignments.
- CONFIG_DFF:
  - Enable CW mode by setting CW_EN to ‘1’
  - Power down all other electronics by writing the rest to ‘0’
- CONFIG_DRV:
  - Select the desired CW operating mode and assign the acoustic channels to the CW lines
- Disable the ASIC clock
- Start CW Transmit/Receive

9.7.2. Handling of Faulty Elements

During the buildup of the acoustic stack, shorts or opens might be introduced to a small number of elements. To avoid shorts between the CW lines, shorted elements can either be connected to the same CW line or left unconnected altogether by setting TX_GND_CW and TX_HVP_CW to ‘1’.

9.7.3. Additional Separation of Transmit and Receive

To reduce any crosstalk that might arise between the transmit and receive planes, it is possible to increase the distance between the transmit and receive planes by connecting neighboring elements to GND by setting TX_GND_CW to ‘1’. Alternatively, the elements can also be connected to HVP by setting TX_HVP_CW to ‘1’.

10. Silicon Errata

CONFIG_DFF.MATRIX_OFFSET_VAL	
Where	Error in setup structure (digital).
Description	The pseudo-random generator generates values between 0-14, but only values between 0-13 are allowed.
Effect	Pseudo-random generator cannot be used.
Workaround	None, write the offset value directly.
Metal fixable	Maybe, low chances.

REG_CERXTXCONF.TX_AppByVal	
Where	Error in setup structure (digital).

Description	`TXDELAY_CECONF.MinVal <= unsigned(reg_cerxtxconf(39 downto 32));`
	`TXDELAY_CECONF.MaxVal <= unsigned(reg_cerxtxconf(39 downto 32));`
	
	Pointing to the same registers.

Effect	`TX_AppByVal` not useable, elements cannot be enabled by value.
Workaround	None, use other aperture definition.
Metal fixable	Maybe, low chances.

CONFIG_DFF. RX_ALWAYS_EN, CONFIG_DFF.ANALOG_RESET_ATNOTRX after TX	
Where	Error in interaction between the periphery and the core (digital).
Description	The receive logic is looking for the end of transmission phase. After the TX phase, if `TX_NRESET` is reset in the periphery with the TX_{CLK}, `S_RX_REG` is reset in the group logic, which disables the RX.
Effect	No receive after transmit without workaround.
Workaround	1. Set `REG_CONFIG.BeamContTx` to '1'. (Can be clocked down after transmit to save power.) The power consumption is higher and this causes large pattern (clk signals) in the dark measurements. 2. Set `RX_ALWAYS_EN` to '1'. Glitch in the nearfield expected, when switching from RX to TX. Set `ANALOG_RESET_ATNOTRX` to '0'. Alternatively, set `REG_CONFIG.BeamContTx` to 1 (needed for pulse cancellation or precharging).
Metal fixable	Maybe, investigation required.

REG_CERXTXCONF aperture changes	
Where	Aperture settings do not work as intended.
Description	Xgroup is calculated in different pipeline stage than ExtrapolElement. Additionally, X/Y are swapped within a group.
Effect	The first 7 elements (Elements 0-6) of a group are enabled in the previous group.
Workaround	Use RX/TX_DISABLE from CWSEL bits <1:0>. Element wise RX/TX disable. See Section 7.2.5.
Metal fixable	No.

SELECT_DFF	
Where	Writing partially using SELECT_DFF does "not fully" work as intended.
Description	For groups on the far side of the ASIC groups 15,31,47,62, 63 there was a functional problem observed not taking all data using partial write and Conditional Write. The error indicates a long path timing violation. From P&R timing analysis it is seen that in WC (125C,1.62V) from group input to output CONF_CLK is delayed by 320ps, while CONF_DIN is delayed by 1300ps. The reserved setup time of 10ns is violated in this case from group 10 on, so the observed error appears to be reasonable. To confirm this, ASIC speed has been reduced, and the error disappeared.
Effect	Due to this silicon bug writing using the SELECT_DFF and Conditional Write does NOT work as intended.
Workaround	1. Use Conditional Write as it is for (groups %16 <= 8) | 8 <14. 2. Write at lower ASIC speeds (e.g. 50MHz, special FPGA code required) 3. Write twice at 100MHz (preferred) a. Write first to first row of groups, and use them as buffer, second write has a setup time 20ns
Metal fixable	Some chances to be fixed by a metal redesign.

11. Document Revisions

Revision	Change description
A, B	Major update
C	Textual improvement in Matlab example code
D	6.5.3 Long Commands: WriteChainMatrix command length 3813 send bytes instead of 1909; WriteChainCondxx, WriteChainCONFIG send bytes corrected; ReadRegisterBIST, ReadRegisterCERXTXCONF return bytes corrected WriteChainCONFIG -> WriteChainCondCONFIG 4.2 Recommended Operating Conditions: HVP is lower for plane wave imaging 9.2 Global Setup: Recommended starting values changed 7.1.7 REG_CONFIG: removed BeamTxAutoStop 7.2 Core Registers: Table 23, *WriteChainCond explained 7.2.1 SELECT_DFF updated 7.2.5 CWSEL_DFF updated 7.2.6 XMITWF_DFF updated 7.2.7 CONFIG_DFF updated 9.6 Pulsed Doppler Mode: setting for MATRIX_OFFSET_VAL changed 10 Silicon Errata: REG_CERXTXCONF aperture changes Appendix A: CONFIG_DRV positions added
E	7.2.7 CONFIG_DFF: ANALOG_RESET_ATNOTRX must be set to 0 (silicon bug) 8.3.5, 8.3.6, 8.3.7 (delay compression) updated 10 Silicon Errata: CONFIG_DFF. ANALOG_RESET_ATNOTRX added Appendix B (MATLAB scripts) updated
F	6.5.1 Table 11: NOP must be full byte 0x00 9.7 Footnote: use left half for receive, right half for transmit 10 Silicon Errata: SELECT_DFF added B.2 Bug fix in UncompressRxDyn
G	7.1.5 REG_RXDYN updated B.2. Rx decompression: CompressTaylorRxDyn, UncompressRxDyn updated
H	7.1.1 REG_BEAMTIMING: TGCIncrement=1 is invalid 7.2.6 XMITWF_DFF: TGC_GAIN <39:32> added 9.7 Footnote (receive, transmit interchangeable) moved to main text
I	4.2 Recommended Operation Conditions: minimum HVP voltage is 8 Volt. 7.1.1 REG_BEAMTIMING: if ExtRxSetupTime='1', REG_CONFIG.BeamWaitRun=1 does not work. 7.1.6 REG_CERXTXCONF does not work as intended.
J	4.2 Recommended Operation Conditions: max. HVP current 10 mA, max. HVP power 500 mW 7.2.6 XMIT_WF: Power consumption for TX_PATTERNREP=63 is high. 8.3.2 Fuse Memory Table updated: added 1 extra bit to the Y-parameter. 8.3.5 TX Delay Compression: offset of element delays is 16. 8.3.6 RX Delay Compression: offset of element delays is 14. 8.3.7 Dynamic Receive: original dynamic delays added. 9.5 Pulsed Mode: updated description. Appendix B: Scripts updated, example for dynamic Rx compression added. Throughout document: *cwater* replaced by *cSound*
K	4.5 Power consumption: digital power consumption updated 4.5 Power consumption: unique polar -> tripolar 4.7 Decoupling capacitors: section added 6.5.3 Long Commands: WriteChainCondCONFIG length corrected 7.1.1 REG_BEAMTIMING: if ExtTxRunTime='1', REG_CONFIG.BeamWaitRun=1 does not work. 7.2.7 CONFIG_DFF: ODRV_EN and RX_ALWAYS_EN updated 7.3.1 CONFIG_DRV: DRV_ENABLE updated
L	3.2 Pin descriptions: VDD10, removed "for CW-mode" 7.2.2 TXDELAY_DFF: Element is disabled by settings 30 and 31 7.2.6 XMITWF_DFF: If TGC gain overwrite is used, do not send XMITWF_DFF on every beam 7.2.6 XMITWF_DFF.TX_CLAMPTIME must be >=4

	7.2.7 CONFIG_DFF.RX_ALWAYS_EN: must be 0 if precharging is used 7.2.7 ISEL_DCGND must be >=1 8.2.5.1 Low pas filtering added: receive signal filtered by a sinc function 8.3.2 Fuses: Fuse memory updated 10 Silicon Errata: Workaround for CONFIG_DFF.RX_ALWAYS_EN updated

A. C++ code examples

```
const int ConfigDRVPosition[] = {23, 21, 31, 30, 29, 20, 22, 18, 28, 27, 26, 17, 19, 16, 25, 24, 15, 6, 7, 4, 14, 13, 12, 3, 5, 1, 11, 10, 9, 0, 2, 8, 40, 33, 32,
42, 41, 43, 34, 36, 35, 45, 44, 46, 37, 39, 38, 56, 47, 57, 48, 50, 49, 59, 58, 60, 51, 53, 52, 62, 61, 63, 54, 55 };

  #define CONST_NR_GROUPS_X          (16)
  #define CONST_NR_GROUPS_Y          ( 4)
  #define CONST_NR_GROUPS            (CONST_NR_GROUPS_X * CONST_NR_GROUPS_Y)
  #define CONST_NR_TXELEMENTS        (16)
  #define CONST_NR_RXELEMENTS        (16)
  #define CONST_NR_ASICELEMENTS      (CONST_NR_GROUPS * 16)
  #define CONST_TX_WAVEFORM_DELAYSTAGES  (15)
  #define CONST_ELEMENT_DELAY_STAGES    (14)
  #define CONST_TX_GROUPDELAY_BITWIDTH   (10)
  #define CONST_ELEMENTDELAY_BITWIDTH    ( 5)
  #define CONST_ELEMENTDYNPHASE_BITWIDTH ( 3)
  #define CONST_RXELEMENTDELAY_BITWIDTH   (CONST_ELEMENTDELAY_BITWIDTH + CONST_ELEMENTDYNPHASE_BITWIDTH)
  #define CONST_NR_RXCLKS            ( 8)

  struct ConfigDRV_t
  {
    unsigned short Enable   : 1;
    unsigned short BiasSel  : 3;
    unsigned short SelInCW  : 4;
    unsigned short SelOutCW : 4;
    unsigned short OutCon   : 1;
    unsigned short SelInNoCW : 1;
    unsigned short ResEn    : 1;
    unsigned short FFen     : 1;
  };
  struct ConfigDRVAll_t
  {
    struct ConfigDRV_t Driver[CONST_NR_GROUPS_Y*CONST_NR_GROUPS_X];
  };

#define CONFIGDRV_REG_SIZE (sizeof(struct ConfigDRVAll_t))
```

A.1. Scramble core registers

```
void asic_driver::ScrambleToSend(unsigned char *dest, unsigned char *DataStuffed, unsigned int BytesPerGroup, unsigned int NrXGroups)
{
  for (unsigned int x=0;x<NrXGroups;x++)
  {
    for (unsigned int bytes=BytesPerGroup; bytes-->0; )
    {
      unsigned char byte[4] = { DataStuffed[0 * NrXGroups*BytesPerGroup + ((NrXGroups-1)-x) * BytesPerGroup + bytes],
                    DataStuffed[1 * NrXGroups*BytesPerGroup + ((NrXGroups-1)-x) * BytesPerGroup + bytes],
                    DataStuffed[2 * NrXGroups*BytesPerGroup + ((NrXGroups-1)-x) * BytesPerGroup + bytes],
                    DataStuffed[3 * NrXGroups*BytesPerGroup + ((NrXGroups-1)-x) * BytesPerGroup + bytes] };
      *dest++ = ((byte[3] & 0xC0)>>0) + ((byte[2] & 0xC0)>>2) + ((byte[1] & 0xC0)>>4) + ((byte[0] & 0xC0)>>6);
      *dest++ = ((byte[3] & 0x30)<<2) + ((byte[2] & 0x30)<<0) + ((byte[1] & 0x30)>>2) + ((byte[0] & 0x30)>>4);
      *dest++ = ((byte[3] & 0x0C)<<4) + ((byte[2] & 0x0C)<<2) + ((byte[1] & 0x0C)<<0) + ((byte[0] & 0x0C)>>2);
      *dest++ = ((byte[3] & 0x03)<<6) + ((byte[2] & 0x03)<<4) + ((byte[1] & 0x03)<<2) + ((byte[0] & 0x03)<<0);
    }
  }
}
```

A.2. Descramble core registers

```
void asic_driver::DescrambleFromReceive(void* DataStuffed, unsigned char *source, unsigned int BytesPerGroup)
{
  unsigned char *DataStuffed1D_ptr = (unsigned char *) DataStuffed;

  for (unsigned int x=CONST_NR_GROUPS_X;x-->0;)
  {
    for (unsigned int bytes=BytesPerGroup; bytes-->0; )
    {
```

```
        for (unsigned int bits=4; bits-->0;)
        {
          unsigned char receivebyte = *source++;
          for (unsigned int y=0;y<CONST_NR_GROUPS_Y;y++)
          {
            DataStuffed1D_ptr[y * CONST_NR_GROUPS_X * BytesPerGroup + x * BytesPerGroup + bytes] |= ((receivebyte >> (y*2)) & 0x03) <<
(2*bits);
          }
        }
      }
    }
  }
}
```

A.3.Scramble CONFIG_DRV

```
int asic_driver::SendConfigDRV(struct ConfigDRVAll_t *ConfigDRV)
{
  struct ConfigDRVAll_t ConfigDRV_Scrambled;
  unsigned char* scrambled_c = (unsigned char*) &ConfigDRV_Scrambled;
  unsigned char*  original_c = (unsigned char*) ConfigDRV;
  my_protocol->SetDescription((char*)"SendConfigDRV");

  for (int groups = CONST_NR_GROUPS/2-1; groups >= 0; groups--)
  {
    //lower CONST_NR_GROUPS/2 are stored in low 4-bits, higher CONST_NR_GROUPS/2 in high 4-bits
    for (int bytesingroup = sizeof(struct ConfigDRV_t)-1; bytesingroup >= 0 ; bytesingroup--)
    {
      *scrambled_c    = ((original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups              ]]>>4) & 0x0F);
      *scrambled_c++ += ((original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups + CONST_NR_GROUPS/2]]>>0) &
0xF0);
      *scrambled_c    = ((original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups              ]]>>0) & 0x0F);
      *scrambled_c++ += ((original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups + CONST_NR_GROUPS/2]]<<4) &
0xF0);
    }
  }
  return WriteRegister(&ConfigDRV_Scrambled, CONFIGDRV_REG_SIZE, WriteChainDRV);
}
```

A.4.Descramble CONFIG_DRV

```
int asic_driver::ReadConfigDRV(struct ConfigDRVAll_t *ConfigDRV)
{
  struct ConfigDRVAll_t ConfigDRV_Scrambled;
  unsigned char* scrambled_c = (unsigned char*) &ConfigDRV_Scrambled;
  unsigned char*  original_c = (unsigned char*) ConfigDRV;
  int return_value;
  my_protocol->SetDescription((char*)"ReadConfigDRV");
  return_value = ReadRegister(&ConfigDRV_Scrambled, CONFIGDRV_REG_SIZE, ReadChainDRV);

  // descramble
  for (int groups = CONST_NR_GROUPS/2-1; groups >= 0; groups--)
  {
    //lower CONST_NR_GROUPS/2 are stored in low 4-bits, higher CONST_NR_GROUPS/2 in high 4-bits
    for (int bytesingroup = sizeof(struct ConfigDRV_t)-1; bytesingroup >= 0 ; bytesingroup--)
    {
      original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups +              0]]  = (*scrambled_c   & 0x0F) << 4;
      original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups + CONST_NR_GROUPS/2]]  = (*scrambled_c++ & 0xF0) << 0;
      original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups +              0]] += (*scrambled_c   & 0x0F) >> 0;
      original_c[bytesingroup+sizeof(struct ConfigDRV_t)*ConfigDRVPosition[groups + CONST_NR_GROUPS/2]] += (*scrambled_c++ & 0xF0) >> 4;
    }
  }
  return return_value;
}
```

B. MATLAB Scripts

B.1. MATLAB script for TX (de)compression

```
function
[cArray,beamOffset_s]=CompressTaylorTx(Fx,Fy,Fz,beamOffset_s,resolution_ns,cSound,nAsics)
    % Calculate coefficients for Tx compression using Taylor series
    % https://en.wikipedia.org/wiki/Taylor_series
    %
    % INPUT:
    %   Fx, Fy, Fz      : focal point in space (meters). Size 1xN -> if N>1, mutiple sets of
c are calculated
    %   beamOffset_s    : delay offset in seconds (0=calculate automatically). Put into
cArray(0),  size 1xN
    %                     !!! maximum beamOffset_s is 255*resolution_ns*1e-9 -> use RunTxTime
instead
    %   resolution_ns   : Tx delay resolution (ns) -- default 12.5
    %   cSound          : sound speed of medium (meters/second)
    %   nAsics          : number of ASICs
    % OUTPUT:
    %   cArray          : array of 8 compressed Tx coefficients, size 8xN
    %   beamOffset_s    : delay offset in seconds. Result from calculations, trying to match
input beamOffset_s
    %
    % VERSION history
    %   2019-05-14:
    %       Calculation of multiple sets of focal points
    %   2020-01-23:
    %       beamOffset_s bug fix


    pitchGroupX=4*180e-6;
    pitchGroupY=4*180e-6;
    XMax=7.5; % can be decreased if aperture reduction is used
    YMax=2*nAsics-0.5; % nAsics=1 --> 1.5; =2 -> 3.5; =3 -> 5.5; =4 -> 7.5;
    tres=resolution_ns*1e-9;
    nSteeringAngles=length(Fx);

    % ASIC spec Section 8.3.5 -> ratio between L lookup table and real world group
coordinates
    L1_factor = 4; %round(7.5/(120/64));
    L2_factor = 64; %round(7.5^2/(56/64));
    L3_factor = 1500; % round(7.5^3/(18/64));
    L4_factor_sq = 32; % round((7.5/(15/(8*sqrt(2))))^2)

    cArray=zeros(8,nSteeringAngles,'int16');

    for iSteering=1:nSteeringAngles
        R=sqrt(Fx(iSteering)^2+Fy(iSteering)^2+Fz(iSteering)^2);
        Fx_R=Fx(iSteering)/R;
        Fy_R=Fy(iSteering)/R;
        c=zeros(1,8);

        % foc_d = sqrt( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 ); % distance from each group (X,Y,Z) to
Focal point (Fx,Fy,Fz); R=sqrt (Fx^2 + Fy^2 + Fz^2); Apex = (0,0,0)
        % delays*cSound = (R-foc_d); % delay for each group [el,geo_focus]

        % Taylor around X=Y=0

        % f_dX = -1 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-0.5) * 0.5 * 2 * (X-Fx);
        % Ax = f_dX(a)/factorial(1) (X-a) = 1/R * R*sin(ax);
        % c1 = Ax/cSound *pitch_group*4; % factor 4 is ratio between L1 and group coordinate
X
        c(2)=Fx_R/cSound *pitchGroupX*L1_factor/tres;
```

```
        % f_dXdX = -1 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-1.5) * -0.5 * (X-Fx) * 2 * (X-Fx) +
...
        %           -1 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-0.5);
        % Axx = f_dXdX(a)/factorial(2) * (X-a)^2 = (1/R^3 * R^2*sin(ax)^2 - 1/R)/2 * X^2 =
...
        %     1/R * (sin(ax)^2 -1)/2 *X^2;
        c(3)=1/R * (Fx_R^2-1)/2 /cSound *pitchGroupX^2*L2_factor/tres;

        % f_dXdXdX = -3 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-2.5) * (X-Fx)^3 + 3 * ( (X-Fx)^2 +
(Y-Fy)^2 + Fz^2 )^(-1.5) * (X-Fx);
        % Axxx = f_dXdXdX(a)/factorial(3) * (X-a)^3 = (3/R^2* (sin(ax))^3 - 3/R^2 *
sin(ax))/6 * X^3 = ...
        %                                                 1/R^2 * ( sin(ax)^3 - sin(ax)) /2 *
X^3;
        c(4)=1/R^2 * (Fx_R^3 - Fx_R)/2 /cSound *pitchGroupX^3* L3_factor/tres;

        % f_dXdY = ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-1.5) * (Y-Fy)*(X-Fx)
        % Axy = f_dXdY(a,a)/fractional(2) * 2 * (X-ax) * (Y-ay) = 1/R *
sind(steering_x)*sind(steering_y)*X*Y
        c(8)=1/R * Fx_R*Fy_R / cSound * pitchGroupX*pitchGroupY * L4_factor_sq/tres;

        % Y
        c(5)=Fy_R/cSound *pitchGroupY*L1_factor/tres; % Y
        c(6)=1/R * (Fy_R^2-1)/2 /cSound *pitchGroupY^2*L2_factor/tres; % Y^2
        c(7)=1/R^2 * (Fy_R^3 - Fy_R)/2 /cSound *pitchGroupY^3* L3_factor/tres; % Y^3

        % --- Start update 2019-09-18
        % c(1): offset in delays -> make sure that all delays are >=1
        c1Min=int16(floor(abs(c(2))/L1_factor * XMax -c(3)/L2_factor * XMax^2
+abs(c(4))/L3_factor * XMax^3 + ...
            abs(c(5))/L1_factor * YMax -c(6)/L2_factor * YMax^2 +abs(c(7))/L3_factor * YMax^3
+ ...
            abs(c(8))/L4_factor_sq * XMax*YMax + 1 +0.5));

        c(1)=int16(floor(beamOffset_s(iSteering)/tres+0.5));

        % clip
        c(1)=min(max(c(1),c1Min),255); % Clip to 0 --> 255
        for i=2:8
            c(i)=min(max(c(i),-128),127); % Clip to -128 --> +127
        end

        c=int16(floor(c+0.5))'; % rounding of the coefficients
        beamOffset_s(iSteering)=(double(c(1)) + 15)*tres; % 2020-01-23: If all coefficients
are zero, uncompressed Tx delays=15 !
        % --- End update 2019-09-18

        % Store results in array
        cArray(:,iSteering)=c;

    end % iSteering

end


function [txDelaysXY,txDelays]=UncompressTx(cArray,nAsics,iAsic)
    % Uncompress Tx delay coefficients
    %
    % INPUT:
    %   cArray  : array of 8 compressed Tx coefficients, size 8xN
    %   nAsics  : number of USX chips
    %   iAsic   : ASIC number (0--3), -1 = all ASICs
    %
    % OUTPUT:
    %   txDelaysXY : delays per element [x,y] in resolution units
```

```
    %   txDelays    : delay [el,group X, group Y] in resolution units, where el=17 is group
delay


    nGroupsX=16;
    nGroupsY=4; % for 1 ASIC

    if iAsic>=0
        iAsicInd=iAsic;
    else
        iAsicInd=0:nAsics-1;
    end

    if length(cArray)~=8
        error('Length of cArray must be 8.');
    end

    txDelaysXY=zeros(nGroupsX*4,nGroupsY*4*length(iAsicInd));
    txDelays=zeros(17,nGroupsX,nGroupsY*length(iAsicInd));
    for i=1:length(iAsicInd)
        iAsic=iAsicInd(i);
        group_offset = 4 - nAsics + 2*iAsic;
        [txDelays_act,txDelaysXY_act] =
TxCompression.DelayUncompressionTxAsic(cArray,group_offset);
        txDelaysXY(:,(1:16) + (i-1)*nGroupsY*4) = txDelaysXY_act;
        txDelays(:,:,(1:4) + (i-1)*nGroupsY)=txDelays_act;
    end

    % Plotting:
    % figure;colormap(jet(256));imagesc(txDelaysXY')
end

function a=TruncateBits(a,nBits)
    a=floor(a);
    a_bin=dec2bin(a);
    l_a_bin=length(a_bin);
    if l_a_bin>nBits % Truncate to 9 bits (unsigned)
        a_bin=a_bin((l_a_bin-nBits+1):l_a_bin);
        a=bin2dec(a_bin);
    end
end

function [txDelays,txDelaysXY] = DelayUncompressionTxAsic(cArray,group_offset)
    % Uncompression of Tx delay coefficients for one USX chip
    %
    % INPUT:
    %   cArray          : array of 8 compressed Tx coefficients, size 8xN
    %   group_offset    : y group offset (see ASIC spec)
    %
    % OUTPUT:
    %   txDelays        : delay [el,group x,group y] in resolution units
    %   txDelaysXY      : delay [x,y] in resolution units

    cArray=double(cArray); % For calculation precision 2020-02-03

    % Coordinates for groups
    y_group_offset = 2*group_offset;

    % Coordinates for elements
    x_local = [-2, -2, -2, -2, -1, -1, -1, -1, 0, 0, 0, 0, 1, 1, 1, 1]; % Local positions,
offset by -0.5
    y_local = [-2, -1, 0, 1, -2, -1, 0, 1, -2, -1, 0, 1, -2, -1, 0, 1]; % Local positions,
offset by -0.5

    % Tables
```

```
    table_linearfs = [-120, -104, -88, -72, -56, -40, -24, -8, 8, 24, 40, 56, 72, 88, 104,
120];
    table_square   = [56, 42, 30, 20, 12, 6, 2, 0, 0, 2, 6, 12, 20, 30, 42, 56];
    table_qube     = [-18, -12, -7, -4, -2, -1, 0, 0, 0, 0, 1, 2, 4, 7, 12, 18];
    table_linearxy = [-15, -13, -11, -9, -7, -5, -3, -1, 1, 3, 5, 7, 9, 11, 13, 15];


    ELEMENT_DELAY_FP=3;
    CENTER_DELAY_FP=3;
    CONST_NR_GROUPS_X=16;
    CONST_NR_GROUPS_Y=4;
    CONST_NR_GROUPS=(CONST_NR_GROUPS_X * CONST_NR_GROUPS_Y);
    CONST_NR_TXELEMENTS=16;

    % Init
    txDelays=zeros(CONST_NR_TXELEMENTS+1,CONST_NR_GROUPS_X,CONST_NR_GROUPS_Y); % last
'element' within a group is the group delay
    txdelay_user=zeros(1,CONST_NR_GROUPS*CONST_NR_TXELEMENTS); % [ns]
    txDelaysXY=zeros(CONST_NR_GROUPS_X*4,CONST_NR_GROUPS_Y*4);

    % Calculate delays
    for y=0:CONST_NR_GROUPS_Y-1
        for x=0:CONST_NR_GROUPS_X-1
            y0 = y+y_group_offset;
            d_group  = floor(cArray(1)*2^CENTER_DELAY_FP);
            d_group = d_group + floor((cArray(5) * table_linearfs(y0+1))/2^CENTER_DELAY_FP);
% due to table_div=64: C4 is bit shifted <<center_delay_fp, thus effectively the same as C0
            d_group = d_group + floor((cArray(6) * table_square(y0+1))/2^CENTER_DELAY_FP);
            d_group = d_group + floor((cArray(7) * table_qube(y0+1))/2^CENTER_DELAY_FP);
            d_group = d_group + floor((cArray(2) * table_linearfs(x+1))/2^CENTER_DELAY_FP);
            d_group = d_group + floor((cArray(3) * table_square(x+1))/2^CENTER_DELAY_FP);
            d_group = d_group + floor((cArray(4) * table_qube(x+1))/2^CENTER_DELAY_FP);
            d_group = d_group + floor((cArray(8) * table_linearxy(x+1) * table_linearxy(y0+1)
)/2^(CENTER_DELAY_FP+1)); % table_linearxy_div=128 -> thus an additional >>1 needed
            if (d_group < 0)
                d_group = 0;
            end
            d_group = floor (d_group/2^CENTER_DELAY_FP); % Includes LSB, which is used for
element_delay
            d_group=TxCompression.TruncateBits(d_group,9); % Truncate to 9 bits (unsigned)

            dY       = cArray(5)*2^(CENTER_DELAY_FP+1);
            dY       = dY + floor((cArray(6) * table_linearfs(y0+1))/2^CENTER_DELAY_FP);
            dY       = dY + floor((cArray(7) * table_square(y0+1))/2^CENTER_DELAY_FP);
            dY       = dY + floor((cArray(8) * table_linearfs(x+1))/2^CENTER_DELAY_FP);
            dY       = floor(dY/2^(CENTER_DELAY_FP+1));
            dY=min(max(dY,-128),127);

            dX       = (cArray(2)*2^(CENTER_DELAY_FP+1));
            dX       = dX + floor((cArray(3) * table_linearfs(x+1))/2^CENTER_DELAY_FP);
            dX       = dX + floor((cArray(4) * table_square(x+1))/2^CENTER_DELAY_FP);
            dX       = dX + floor((cArray(8) * table_linearfs(y0+1))/2^CENTER_DELAY_FP);
            dX       = floor(dX/2^(CENTER_DELAY_FP+1));
            dX=min(max(dX,-128),127);

            txDelays(CONST_NR_TXELEMENTS+1,x+1,y+1) = floor(d_group/2);
            for ele=0:CONST_NR_TXELEMENTS-1 % for each element within group
                % element_delay;             // Local Element delay as 5.3 fixed point
(ELEMENT_DELAY_FP)
                ind = (y*CONST_NR_GROUPS_X + x)*CONST_NR_TXELEMENTS + ele + 1;
                element_delay = (16*2^ELEMENT_DELAY_FP);  % Center delay is 16
                if mod(d_group,2)==0; element_delay = element_delay - (1*2^ELEMENT_DELAY_FP);
end % take account for centerdelay LSB if even % 2020-02-03 Corrected
                element_delay = element_delay + floor(dX/2^2) + floor((x_local(ele+1) *
dX)/2); % 2020-02-03 Corrected
```

```
                element_delay = element_delay + floor(dY/2^2) + floor((y_local(ele+1) *
dY)/2); % 2020-02-03 Corrected
                element_delay = floor(element_delay/2^ELEMENT_DELAY_FP);
                if (element_delay >= 30); element_delay = 255; end % Check if range is
usable(0 - 29 are valid)
                txdelay_user(ind) = floor(d_group/2)*2 + element_delay; % remove Bit0 from
d_group
                txDelays(ele+1,x+1,y+1) = element_delay; % Element Delay
                txDelaysXY(x*4+floor(ele/4)+1,y*4+mod(ele,4)+1)=txdelay_user(ind);
            end
        end
    end
end
```

B.2.MATLAB script for RX (de)compression

```
function cArray=CompressTaylorRx(Fx,Fy,Fz,resolution_ns,cSound,nAsics)
    % Calculate coefficients for Tx compression using Taylor series
    % https://en.wikipedia.org/wiki/Taylor_series
    %
    % INPUT:
    %   Fx, Fy, Fz      : focal point in space (meters). Size 1xN -> if N>1, multiple sets of
c are calculated
    %   resolution_ns   : Tx delay resolution (ns) -- default 20
    %   cSound          : sound speed of medium (meters/second)
    %   nAsics          : number of ASICs
    %
    % OUTPUT:
    %   cArray          : array of 10 compressed Tx coefficients, size 10xN
    %
    % See ASIC spec Section 8.3.6

    pitchElX=180e-6;
    pitchElY=180e-6;
    pitchGroupX=pitchElX*4;
    pitchGroupY=pitchElY*4;
    tres=resolution_ns*1e-9;
    nSteeringAngles=length(Fx);

    % ASIC spec Section 8.3.5 -> ratio between L lookup table and real world group
coordinates
    L0_factor = 32*4; % round(1/(1/32))*4; *4 is from c9/4
    L1_factor = 256*4; %round(7.5/(30/1024));  *4 is from c9/4
    L2_factor = 2048*4; %round(7.5^2/(28/1024)); *4 is from c9/4
    c7_c8_factor = 16;

    % Determine the maximum Y coordinate as function of nAsics
    L1_max=30;
    L2_max=28;
    switch nAsics
        case 1; L1_max=6;    L2_max=1;
        case 2; L1_max=14;   L2_max=6;
        case 3; L1_max=22;   L2_max=15;
        case 4; L1_max=30;   L2_max=28;
    end

    cArray=zeros(10,nSteeringAngles,'int16');

    for iSteering=1:nSteeringAngles
        R=sqrt(Fx(iSteering)^2+Fy(iSteering)^2+Fz(iSteering)^2);
        Fx_R=Fx(iSteering)/R;
        Fy_R=Fy(iSteering)/R;
        c=zeros(1,10);

        c(8)=Fx_R/cSound*pitchElX*c7_c8_factor/tres;
        c(9)=Fy_R/cSound*pitchElY*c7_c8_factor/tres;

        % Taylor around X=Y=0

        % dX
        % F = -1 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-0.5) * 0.5 * 2 * (X-Fx);
        % A = F(a) (X-a) = 1/R * R*sin(ax);
        c(1)=(Fx_R/cSound *pitchElX/tres - floor(c(8)+0.5)/c7_c8_factor) *L0_factor; %
Resolution of dX is 4 times higher (see c9/4 in formula)

        % First derivative
        % Fx = -1 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-1.5) * -0.5 * (X-Fx) * 2 * (X-Fx) + ...
        %         -1 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-0.5);
        % Ax = Fx(a)/factorial(1) * (X-a)^2 = (1/R^3 * R^2*sin(ax)^2 - 1/R) * X^2 = ...
        %     1/R * (sin(ax)^2 -1) *X^2;
```

```
        c(2)=1/R * (Fx_R^2-1) /cSound *pitchElX*pitchGroupX*L1_factor/tres;

        % Fxx = -3 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-2.5) * (X-Fx)^3 + 3 * ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-1.5) * (X-Fx);
        % Axx = f_dXdXdX(a)/factorial(2) * (X-a)^3 = (3/R^2* (sin(ax))^3 - 3/R^2 * sin(ax))/2 * X^3 = ...
        %                                                3/R^2 * ( sin(ax)^3 - sin(ax)) /2 * X^3;
        c(3)=3/R^2 * (Fx_R^3 - Fx_R)/2 /cSound *pitchElX*pitchGroupX^2* L2_factor/tres;

        % First derivative to X -> dF/dX
        % d dY / dx = ( (X-Fx)^2 + (Y-Fy)^2 + Fz^2 )^(-1.5) * (Y-Fy)*(X-Fx)
        % A = F(y)x(a,a) * (X-ay) = 1/R * sind(steering_x)*sind(steering_y)*X
        c(7)=1/R * Fx_R*Fy_R / cSound * pitchElY*pitchGroupX * L1_factor/tres;

        % Y
        c(4)=(Fy_R/cSound *pitchElY/tres -floor(c(9)+0.5)/c7_c8_factor) *L0_factor; % Y
        c(5)=1/R * (Fy_R^2-1) /cSound *pitchElY*pitchGroupY*L1_factor/tres; % Y^2
        c(6)=3/R^2 * (Fy_R^3 - Fy_R)/2 /cSound *pitchElY*pitchGroupY^2* L2_factor/tres; % Y^3

        % assignment_scaler, distance between the assignment steps in 5 ns units -- use 8 for the moment if dynamic delays (=40 ns)
        % Optimize c9 for maximum span
        % With c9, the assignment (defined by dX*(xel-1.5) dY*(yel-1.5) can be scaled.
        % Constraints:
        %   assignment must be 0..7
        %   c(1:7) must not be clipped, e.g. -128 .. +127
        c9_trial=[1,4,8,16];
        for i=1:length(c9_trial)
            c_try=c/c9_trial(i);
            if min(c_try(1:7))<-128 || max(c_try(1:7))>127
                continue; % one of the constants is outside boundary -> go to next c9
            end
            [Amin,Amax]=GetAssignmentMinMax(c_try,L1_max,L2_max);
            if floor(Amin+0.5)>=0 && floor(Amax+0.5)<=7
                break;
            end
        end
        c(10)=c9_trial(i);

        for i=1:7
            c(i)=c(i)/c(10); % compensate the others for this c9
        end

        % clip
        for i=1:9
            c(i)=min(max(c(i),-128),127); % Clip to -128 --> +127
        end
        c(10)=min(max(c(10),0),255); % Clip to 0 --> 255

        c=int16(floor(c+0.5))'; % rounding of the coefficients

        cArray(:,iSteering)=c;
    end % iSteering

    % ----------

    function [Amin,Amax]=GetAssignmentMinMax(c,L1_max,L2_max) % for 2 ASICs
        assignment= c(1)/32+c(2)*[-30,30;-30,30]/1024 +c(3)*28/1024 + ...
                    c(4)/32+c(5)*[-1,-1;1,1]*L1_max/1024+c(6)*L2_max/1024+c(7)*[-30,30;-30,30]/1024 + ...
                    3.5;
        Amin=min(assignment(:));
        Amax=max(assignment(:));
    end
```

```
end


function
[cArray,slope,duration,masterCurve]=CompressTaylorRxDyn(steering_x,steering_y,nAsics,cSound,r
unRxTime)
    % Calculates Rx compression coefficients, dynamic Rx slope and duration
    %
    % INPUT:
    %   steering_x  : x steering angle in degrees
    %   steering_y  : y steering angle in degrees
    %   nAsics      : number of ASICs
    %   cSound      : sound speed (1540 m/s)
    %   runRxTime   : RunRxTime setting of ASIC, at least 29, max 255
    %
    % OUTPUT:
    %   cArray      : c0..c9 of REG_RXDELAY
    %   slope       : slope of dynamic Rx curve (REG_RXDYN; 8 values)
    %   duration    : duration of dynamic Rx curve (REG_RXDYN; 8 values)
    %   masterCurve : delay values of the curve after time 0 and cumsum(tDuration) (9 values)
-- for debugging purposes

    % Constants
    Rstart=10e-3; % Depth (meters) where dynamic Rx starts
    Rmax=100e-3;  % Depth (meters) where dynamic Rx ends -> hereafter, delays stay constant
    asicTimeStep=2.56e-6;
    nSteps=8;
    slopeFactor=256;
    dynDelayResolution=5e-9;
    scalerValues=[7,5,3,1,-1,-3,-5,-7];
    pitchElX=180e-6;
    pitchElY=180e-6;
    pitchGroupX=4*pitchElX;
    pitchGroupY=4*pitchElY;
    runRxTimeStep=40e-9;
    microdelay_step_ns=20;

    % Precalculate the coordinates of the elements within the group (unit = element count;
0=center)
    xEl=zeros(1,16);
    yEl=zeros(1,16);
    for iElY=1:4
        for iElX=1:4
            xEl((iElY-1)*4+iElX)=(iElX-2.5);
            yEl((iElY-1)*4+iElX)=(iElY-2.5);
        end
    end

    % Precalculate the coordinates of the groups within the ASIC (unit = group count;
0=center)
    xGroup=zeros(1,64*nAsics);
    yGroup=zeros(1,64*nAsics);
    for iGroupY=1:4*nAsics
        for iGroupX=1:16
            xGroup(iGroupX+(iGroupY-1)*16)=(iGroupX-8.5);
            yGroup(iGroupX+(iGroupY-1)*16)=(iGroupY-2.5-2*(nAsics-1));
        end
    end

    % I. Calculate Rx compression coefficients
    A=sqrt(1-(sind(steering_x)).^2.*(sind(steering_y)).^2);
    Fx0=Rstart.*sind(steering_x).*cosd(steering_y) ./ A;
    Fy0=Rstart.*sind(steering_y).*cosd(steering_x) ./ A;
    Fz0=Rstart.*cosd(steering_x).*cosd(steering_y) ./ A;

cArray=double(RxCompression.CompressTaylorRx(Fx0,Fy0,Fz0,microdelay_step_ns,cSound,nAsics));
```

```
    % II. Microdelays at the start of the dynamic curves, these are
    % thus the initial microdelays
    [~,~,rxDelaysStart,rxPhase]=RxCompression.UncompressRx(cArray,nAsics,-1); % Uncompress
start delays

rxDelaysStart=reshape(rxDelaysStart,size(rxDelaysStart,1),size(rxDelaysStart,2)*size(rxDelays
Start,3)); % [el (ASIC),group]
    rxDelaysStart=rxDelaysStart([1,5,9,13,2,6,10,14,3,7,11,15,4,8,12,16],:); % Notation of
ASIC is xy swapped -> correct this
    rxDelaysStart=(rxDelaysStart-14.5)*microdelay_step_ns*1e-9; % to seconds
    rxPhase=reshape(rxPhase,size(rxPhase,1),size(rxPhase,2)*size(rxPhase,3)); % [el
(ASIC),group]
    rxPhase=rxPhase([1,5,9,13,2,6,10,14,3,7,11,15,4,8,12,16],:); % Notation of ASIC is xy
swapped -> correct this
    delayMicroDeltaAssignment=rxPhase;
    delayMicroStart=rxDelaysStart;

    % III. To calculate the masterCurve, the microdelays are calculated
    % at certain time positions -> called "snapshots"
    % Determine time points for snapshots and store in tCurveSegments
    t0=(Rstart/cSound*2-runRxTime*runRxTimeStep)/asicTimeStep; % in 2.56 µs increments
    tMax=(Rmax/cSound*2-runRxTime*runRxTimeStep)/asicTimeStep; % time axis in 2.56 µs
increments
    if t0>=0.5 % only if t0 is large enough: add additional t0 point
        t=[0, ((0:nSteps-1)/(nSteps-1)*sqrt(tMax-t0)).^2+t0]; % equally space the square root
from startDepthInd to max
    else
        t=((0:nSteps)/(nSteps)*sqrt(tMax-t0)).^2+t0; % equally space the square root from
startDepthInd to max
        t(1)=0;
    end
    duration=t(2:nSteps+1)-t(1:nSteps); % this is the duration for every step
    tCurveSegments=zeros(1,nSteps+1);
    for i=1:nSteps % the duration has to be rounded; error should be propagated
        tDurationNew=min(max(floor(duration(i)),1),15); % should be at least one and max 15
(4-bit)
        durationError=duration(i)-tDurationNew;
        duration(i)=tDurationNew;
        % since a duration error is made, it should be corrected
        if i<nSteps % Correct duration in the next step
            duration(i+1)=duration(i+1)+durationError;
        end
        tCurveSegments(i+1)=tCurveSegments(i)+duration(i);
    end

    % IV. The mastercurve is calculated at the tCurveSegments in the
    % following way.
    % Each time tCurveSegments is converted to focal depth R.
    % The corresponding microdelay (defined by element delays minus
    % group delays) is calculated and stored in delayMicroIdeal.
    % The difference between the delayMicroIdeal and
    % delayMicroStart is called delayMicroDelta.
    % This delayMicroDelta needs to be covered by the masterCurve times the scaler value.
    % The choice of the scaler value (-7..+7) is given by the assignment.
    masterCurve=zeros(1,nSteps+1);
    d=@(R,x,y) -1/cSound* sqrt( (R*sind(steering_x)*cosd(steering_y)/A-x).^2 +
(R*sind(steering_y)*cosd(steering_x)/A-y).^2 + (R*cosd(steering_x)*cosd(steering_y)/A).^2);
    R=(tCurveSegments*asicTimeStep+runRxTime*runRxTimeStep)/2*cSound; % focal depth;
runRxTime is an offset and needs to be added
    for iStep=2:nSteps+1
        % masterCurve(1) is 0, since t<t0 is constant
        % Determine masterCurve value at tCurveSegments(i)
        delayMicroIdeal=zeros(16,64*nAsics);
        for iGroup=1:64*nAsics
            delayGroup=d(R(iStep),xGroup(iGroup)*pitchGroupX,yGroup(iGroup)*pitchGroupY);
```

```
            for iEl=1:16

delayMicroIdeal(iEl,iGroup)=d(R(iStep),xEl(iEl)*pitchElX+xGroup(iGroup)*pitchGroupX,yEl(iEl)*
pitchElY+yGroup(iGroup)*pitchGroupY)-delayGroup; % Micro delay is element delay minus group
delay
            end
        end
        delayMicroDelta=delayMicroIdeal-delayMicroStart;

masterCurve(iStep)=mean(mean(delayMicroDelta./scalerValues(delayMicroDeltaAssignment+1)));
    end

    % The mastercurve is converted to slope segments
    slope=zeros(1,nSteps);
    for iStep=1:nSteps
        % mastercurve in seconds. /dynDelayResolution -> dynamic step units.
        % Slope: dynDelayResolution units (5 ns) / asicTimeStep (2.56 µs)  * slopeFactor
(256)
        slopeAct_SI=(masterCurve(iStep+1)-masterCurve(iStep))/
(duration(iStep)*asicTimeStep); % slope in SI units: delay/time = s/s = -
        slope(iStep)=slopeAct_SI*asicTimeStep*slopeFactor/dynDelayResolution; % slope for
ASIC is in units of 5 ns / (2.56 µs * 256)
    end
    if slope(2)>255 % Slope is 8 bit -> clips at 255. Propagate slope(2) backwards
        slope(1)=(slope(2)-255)*duration(2)/duration(1);
        slope(2)=255;
    end
    delayError=0;
    for iStep=1:nSteps % Propagate slope forward
        slopeWanted=slope(iStep)+delayError/duration(iStep);
        slopeAct=round(max(min(slopeWanted,255),0)); % clip to 0--255 and round
        delayError=(slopeWanted-slopeAct)*duration(iStep); % Rounding or clipping causes an
"error" -> put this into the next slope
        slope(iStep)=slopeAct;
    end
end


function [rxDelaysXY,rxPhaseXY,rxDelays,rxPhase]=UncompressRx(cArray,nAsics,iAsic)
    % Uncompress Rx delay coefficients
    %
    % INPUT:
    %   cArray      : array of 10 compressed Rx coefficients, size 1x10
    %   nAsics      : number of USX chips
    %   iAsic       : ASIC number (0--3), -1 = all ASICs
    %
    % OUTPUT:
    %   rxDelaysXY  : delays per element [x,y] in resolution units
    %   rxDelays    : delay [el,group X, group Y] in resolution units
    %   rxPhaseXY   : phase per element [x,y] in resolution units
    %   rxPhase     : phase [el,group X, group Y] in resolution units

    nGroupsX=16;
    nGroupsY=4;

    if iAsic>=0
        iAsicInd=iAsic;
    else
        iAsicInd=0:nAsics-1;
    end

    if length(cArray)~=10
        error('Length of cArray must be 10.');
    end

    rxDelays=zeros(16,nGroupsX,nGroupsY*length(iAsicInd));
```

```
    rxPhase=zeros(16,nGroupsX,nGroupsY*length(iAsicInd));
    rxDelaysXY=zeros(nGroupsX*4,nGroupsY*4*length(iAsicInd));
    rxPhaseXY=zeros(nGroupsX*4,nGroupsY*4*length(iAsicInd));
    for i=1:length(iAsicInd)
        iAsic=iAsicInd(i);
        group_offset = 4 - nAsics + 2*iAsic;
        [rxPhase_act,rxDelays_act,rxDelaysXY_act,rxPhaseXY_act] =
RxCompression.DelayUncompressionRxAsic(cArray,group_offset);
        rxDelays(:,:,(1:4) + (i-1)*nGroupsY)=rxDelays_act;          % [el,group X, group Y]
        rxPhase(:,:,(1:4) + (i-1)*nGroupsY)=rxPhase_act;            % [el,group X, group Y]
        rxDelaysXY(:,(1:16) + (i-1)*nGroupsY*4) = rxDelaysXY_act;   % [x,y]
        rxPhaseXY(:,(1:16) + (i-1)*nGroupsY*4) = rxPhaseXY_act;     % [x,y]
    end

    % Plotting:
    % figure;colormap(jet(256));imagesc(rxDelaysXY')
end


function
[rxDelaysXY,rxPhaseXY,rxDelays,rxPhase,masterCurve]=UncompressRxDyn(cArray,nAsics,iAsic,runRx
Time,slope,duration,tAxis)
    % Uncompress dynamic Rx curves
    % See ASIC spec 7.1.5, 8.3.6, 8.3.7
    %
    % INPUT:
    %   cArray          : array of 10 compressed Rx coefficients, size 1x10
    %   nAsics          : number of USX chips
    %   iAsic           : ASIC number (0--3), -1 = all ASICs
    %   runRxTime       : see ASIC spec, default 29
    %   slope           : array 1x8 with slope (see ASIC spec)
    %   duration        : array 1x8 with duration (see ASIC spec)
    %   tAxis           : time array 1xN with time in seconds; dynamic delays are calculated
for these times
    %
    % OUTPUT:
    %   rxDelaysXY      : delay per element [x,y] in resolution units (typ. 20 ns)
    %   rxPhaseXY       : phase per element [x,y] (0--7)
    %   rxDelays        : delay [el,group X, group Y] in resolution units (typ. 20 ns)
    %   rxPhase         : phase [el,group X, group Y] (0--7)
    %   masterCurve     : delay values of the curve after time 0 and cumsum(tDuration) (9
values).
    %                       Unit is duration*slope = 2.56µs * 5ns / (2.56µs * 256) = 5ns /
256.

    nGroupsX=16;
    nGroupsY=4;
    nEl=16;
    nCurves=8;
    runRxTimeStep=40e-9;
    ratioDynToStaticDelayRes=0.25; % In units of full Rx delay resolution (full delay res 20
ns, dyn delay res 5 ns)
    nSteps=8;
    asicTimeStep=2.56e-6;
    multiplier= [7,5,3,1,1,3,5,7];
    multisign=  [1,1,1,1,-1,-1,-1,-1];
    tAxisLength=length(tAxis); % Get length of time axis

    if length(cArray)~=10
        error('Length of cArray must be 10.');
    end

    if iAsic>=0
        iAsicInd=iAsic;
    else
        iAsicInd=0:nAsics-1;
```

```
    end


[rxDelaysStartXY,rxPhaseXY,rxDelaysStart,rxPhase]=RxCompression.UncompressRx(cArray,nAsics,iA
sic); % Uncompress start delays

    % Reconstruct masterCurve
    masterCurve=zeros(1,length(tAxis));
    t=[0,cumsum(duration)*asicTimeStep]+runRxTime*runRxTimeStep; % Correct for runRxTime
offset
    delaySum=cumsum(duration.*slope);
    masterCurve(tAxis<t(1))=0; % If time axis contains negative values -> set masterCurve to
zero
    for i=1:nSteps
        ind=tAxis>=t(i) & tAxis<t(i+1);
        masterCurve(ind)=slope(i)*(tAxis(ind)-t(i))/asicTimeStep; % Take integral of slope *
time elapsed for this segment
        if i>1
            masterCurve(ind)=masterCurve(ind)+delaySum(i-1); % masterCurve is present segment
+ all previous segments (duration*slope)
        end
    end
    % Keep constant beyond last point
    ind=find(tAxis>=t(nSteps+1));
    if ~isempty(ind)
        masterCurve(ind)=slope(nSteps)*(tAxis(ind(1))-t(nSteps))/asicTimeStep;
        masterCurve(ind)=masterCurve(ind)+delaySum(nSteps-1);
    end

    % Add the curves to the start delays
    % Start delays have typ. 20 ns resolution, dynamic delays typ.
    % 5 ns resolution. Thus ratioDynToStaticDelays=0.25.
    % Another factor 1/256 must be included (see unit in ASIC spec 7.1.5)
    rxDelays=zeros(nEl*nGroupsX*nGroupsY*length(iAsicInd),tAxisLength);
    rxDelaysXY=zeros(nGroupsX*4*nGroupsY*4*length(iAsicInd),tAxisLength);
    for iCurve=1:nCurves
        for iEl=1:nEl*nGroupsX*nGroupsY*length(iAsicInd)
            if rxPhase(iEl)==iCurve-1 % this curve
                for iTime=1:tAxisLength

rxDelays(iEl,iTime)=rxDelaysStart(iEl)+multisign(iCurve)*floor(multiplier(iCurve)*masterCurve
(iTime)/256)*ratioDynToStaticDelayRes;
                    % Note that dynamic delay unit "multiplier*masterCurve/256" = 0.25*full
delay unit
                end
            end
            if rxPhaseXY(iEl)==iCurve-1 % this curve
                for iTime=1:tAxisLength

rxDelaysXY(iEl,iTime)=rxDelaysStartXY(iEl)+multisign(iCurve)*floor(multiplier(iCurve)*masterC
urve(iTime)/256)*ratioDynToStaticDelayRes;
                    % Note that dynamic delay unit "multiplier*masterCurve/256" = 0.25*full
delay unit
                end
            end
        end
    end
    % Reshape
    rxDelaysXY=reshape(rxDelaysXY,nGroupsX*4,nGroupsY*4*length(iAsicInd),tAxisLength);
    rxDelays=reshape(rxDelays,nEl,nGroupsX,nGroupsY*length(iAsicInd),tAxisLength);
end


function
[rxPhase,rxDelays,rxDelaysXY,rxPhaseXY]=DelayUncompressionRxAsic(cArray,group_offset)
    % Uncompression of compressed coefficients, mimic the ASIC core
```

```
    %
    % INPUT:
    %   cArray          : array of 10 compressed Rx coefficients, size 10xN
    %   group_offset    : y group offset (see ASIC spec: RXDELAY_SHIFT)
    %
    % OUTPUT:
    %   rxphase         : phase [el,group x,group y]
    %   rxDelays        : delay [el,group x,group y] in resolution units
    %   rxDelaysXY      : delay [x,y] in resolution units
    %   rxPhaseXY       : phase [x,y] in resolution units

    % Pars
    Ncurves=8;
    y_group_offset = 2*group_offset;

    % Coordinates for elements
    x_local=[-1.5, -1.5, -1.5, -1.5, -0.5, -0.5, -0.5, -0.5, 0.5, 0.5, 0.5, 0.5, 1.5, 1.5,
1.5, 1.5]-0.5;
    y_local=[-1.5, -0.5, 0.5, 1.5, -1.5, -0.5, 0.5, 1.5, -1.5, -0.5, 0.5, 1.5, -1.5, -0.5,
0.5, 1.5]-0.5;

    % Table
    table_linear=[-30, -26, -22, -18, -14, -10, -6, -2, 2, 6, 10, 14, 18, 22, 26, 30];
    table_square=[28, 21, 15, 10, 6, 3, 1, 0, 0, 1, 3, 6, 10, 15, 21, 28];

    ELEMENT_ASSIGN_FP=2^3;
    ELEMENT_DELAY_FP=2^3;
    CONST_ELEMENT_DELAY_STAGES=14;
    RXDELAY_OFFSET=0;
    CONST_NR_GROUPS_X=16;
    CONST_NR_GROUPS_Y=4;
    CONST_NR_RXELEMENTS=16;

    % Calculate Assignment
    rxPhase=zeros(1,CONST_NR_RXELEMENTS*CONST_NR_GROUPS_X*CONST_NR_GROUPS_Y); % mem alloc
    rxDelays=zeros(1,CONST_NR_RXELEMENTS*CONST_NR_GROUPS_X*CONST_NR_GROUPS_Y); % mem alloc
    rxDelaysXY=zeros(CONST_NR_GROUPS_X*4,CONST_NR_GROUPS_Y*4);
    rxPhaseXY=zeros(CONST_NR_GROUPS_X*4,CONST_NR_GROUPS_Y*4);

    element_delay_base = CONST_ELEMENT_DELAY_STAGES*ELEMENT_DELAY_FP + floor(cArray(8)/4) +
floor(cArray(9)/4) + ELEMENT_DELAY_FP/2;
    for y=0:CONST_NR_GROUPS_Y-1
        for x=0:CONST_NR_GROUPS_X-1
            y0 = y+y_group_offset;
            dX=floor(cArray(1)) + floor(cArray(2)*table_linear(x+1)/32)+
floor(cArray(3)*table_square(x+1)/32);
            dX=floor(dX/2);
            dY=floor(cArray(4)) + floor(cArray(5)*table_linear(y0+1)/32) +
floor(cArray(6)*table_square(y0+1)/32) + floor(cArray(7)*table_linear(x+1)/32);
            dY=floor(dY/2);

            for ele=0:CONST_NR_RXELEMENTS-1 % for each element within group
                ind=((y+1-1)*CONST_NR_GROUPS_X + x+1-1)*16 + ele+1;

                % Assignment
                rxPhase(ind)=floor(8*ELEMENT_ASSIGN_FP) + ... % 0.5 for Centered rounding +
Center delay 3.5 + margin to avoid overflow 4 = 8
                    floor(dX/2^2) + ... % 0.5 for X Center
                    floor ((dX * x_local(ele+1))/2) + ...
                    floor(dY/2^2) + ...
                    floor ((dY * y_local(ele+1))/2);
                rxPhase(ind)=floor((rxPhase(ind) - 4*ELEMENT_ASSIGN_FP)/ELEMENT_ASSIGN_FP);
% -Ncurves/2
                if rxPhase(ind)<0 % Clip
                    rxPhase(ind)=0;
                end
```

```
                if rxPhase(ind)>Ncurves-1
                    rxPhase(ind)=Ncurves-1;
                end

                % Element delay
                rxDelays(ind)=element_delay_base + floor(cArray(8)*x_local(ele+1)/2) +
floor(cArray(9)*y_local(ele+1)/2) + floor((rxPhase(ind)*2+RXDELAY_OFFSET-(Ncurves-
1))*cArray(10)); % units of 2.5 ns
                rxDelays(ind)=floor(rxDelays(ind)/ELEMENT_DELAY_FP); % units of 2.5 ns
                rxDelaysXY(x*4+floor(ele/4)+1,y*4+mod(ele,4)+1)=rxDelays(ind);
                rxPhaseXY(x*4+floor(ele/4)+1,y*4+mod(ele,4)+1)=rxPhase(ind);
            end
        end
    end

    rxPhase=reshape(rxPhase,CONST_NR_RXELEMENTS,CONST_NR_GROUPS_X,CONST_NR_GROUPS_Y);
    rxDelays=reshape(rxDelays,CONST_NR_RXELEMENTS,CONST_NR_GROUPS_X,CONST_NR_GROUPS_Y);
end
```

B.3.Example for dynamic Rx compression

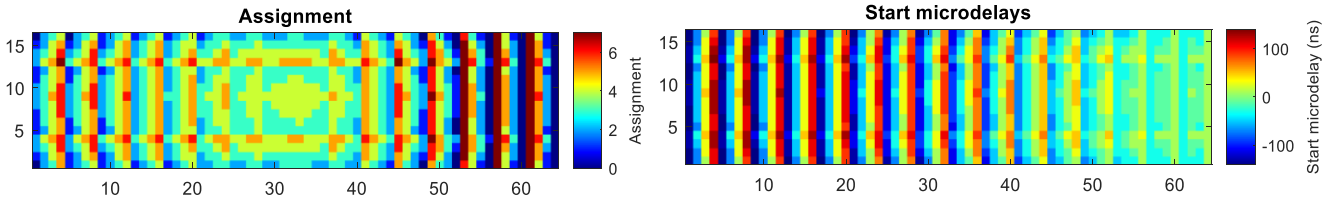
In this section, an example is given for the function `CompressTaylorRxDyn`, steering angle (α_x, α_y)=(30,0), for one ASIC.

Step 1

The Rx compression coefficients are [0 -81 -35 0 -108 0 0 47 0 4].

Step 2

The assignment and start microdelays (at a depth of 10 mm) are given below.

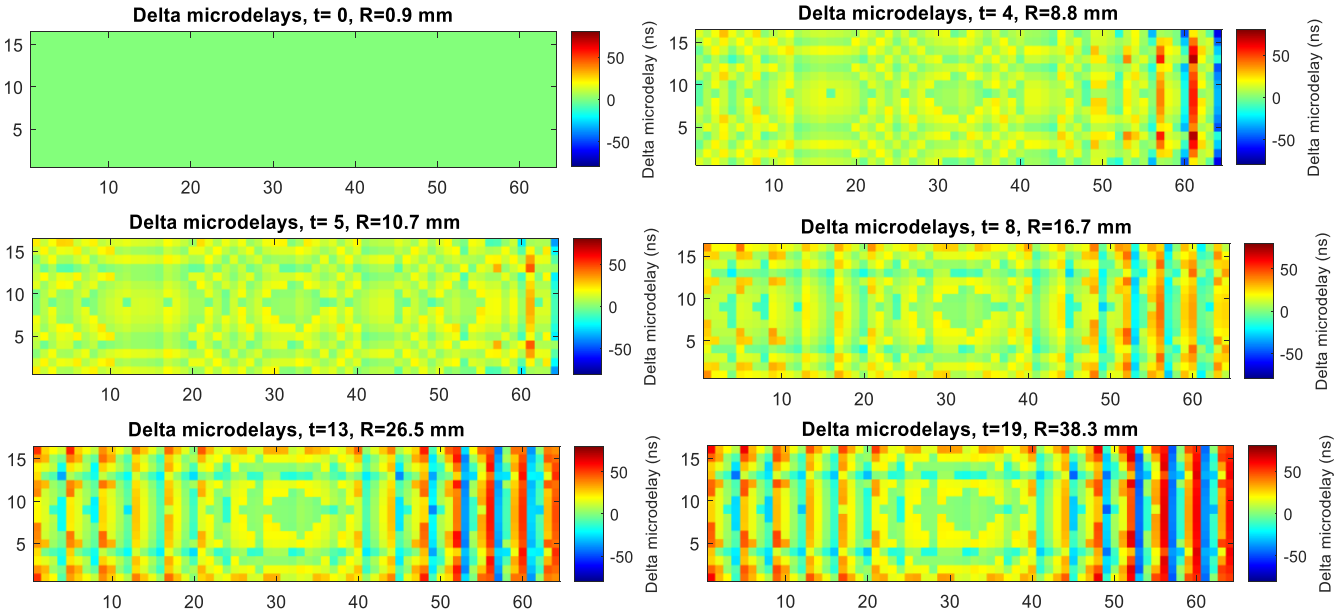


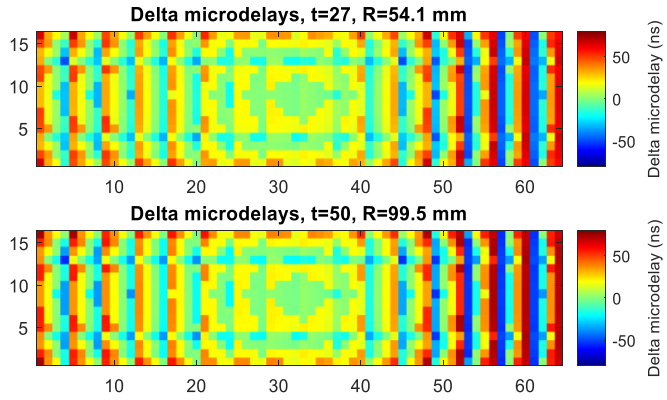
Step 3

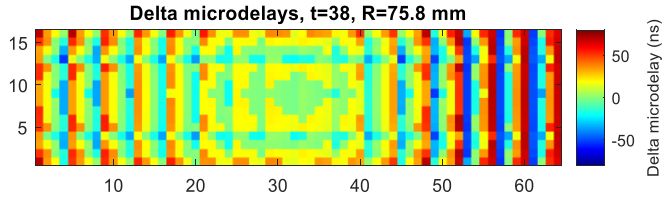
The time positions are [0 4 5 8 13 19 27 38 50].

Step 4

The delta microdelays at the time positions are given below (called “snapshots”).

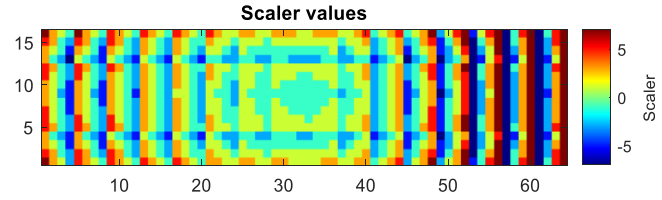


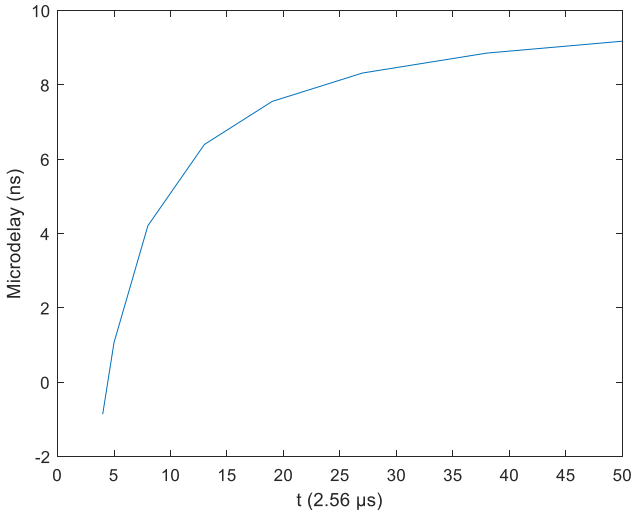




The master curve is obtained by:

1. For each time position, dividing the delta micro delays by the scaler values
2. For each time position, taking the average over the whole ASIC.
3. Determining the slope for each time segment.





Slope = [0 54 54 22 10 5 2 2].