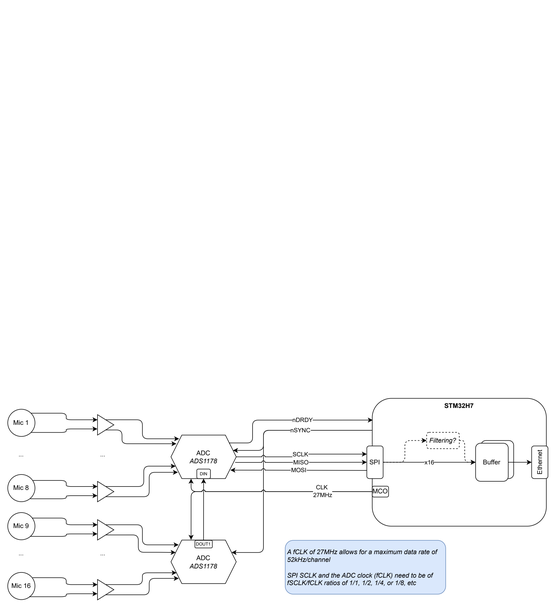
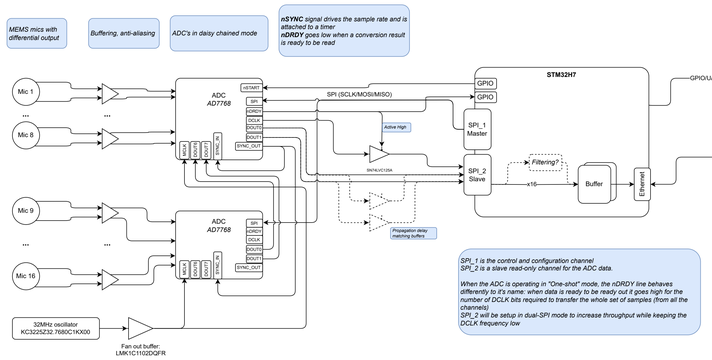
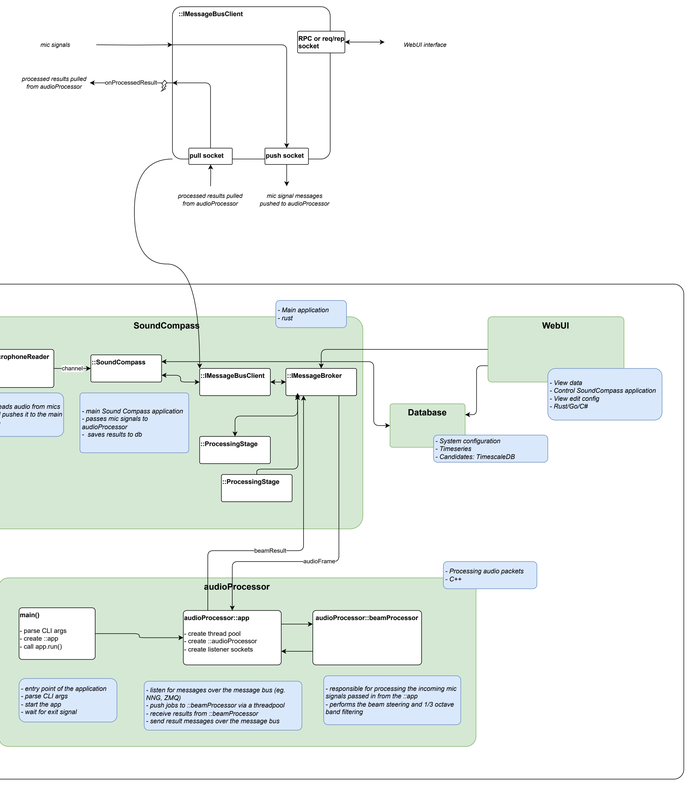
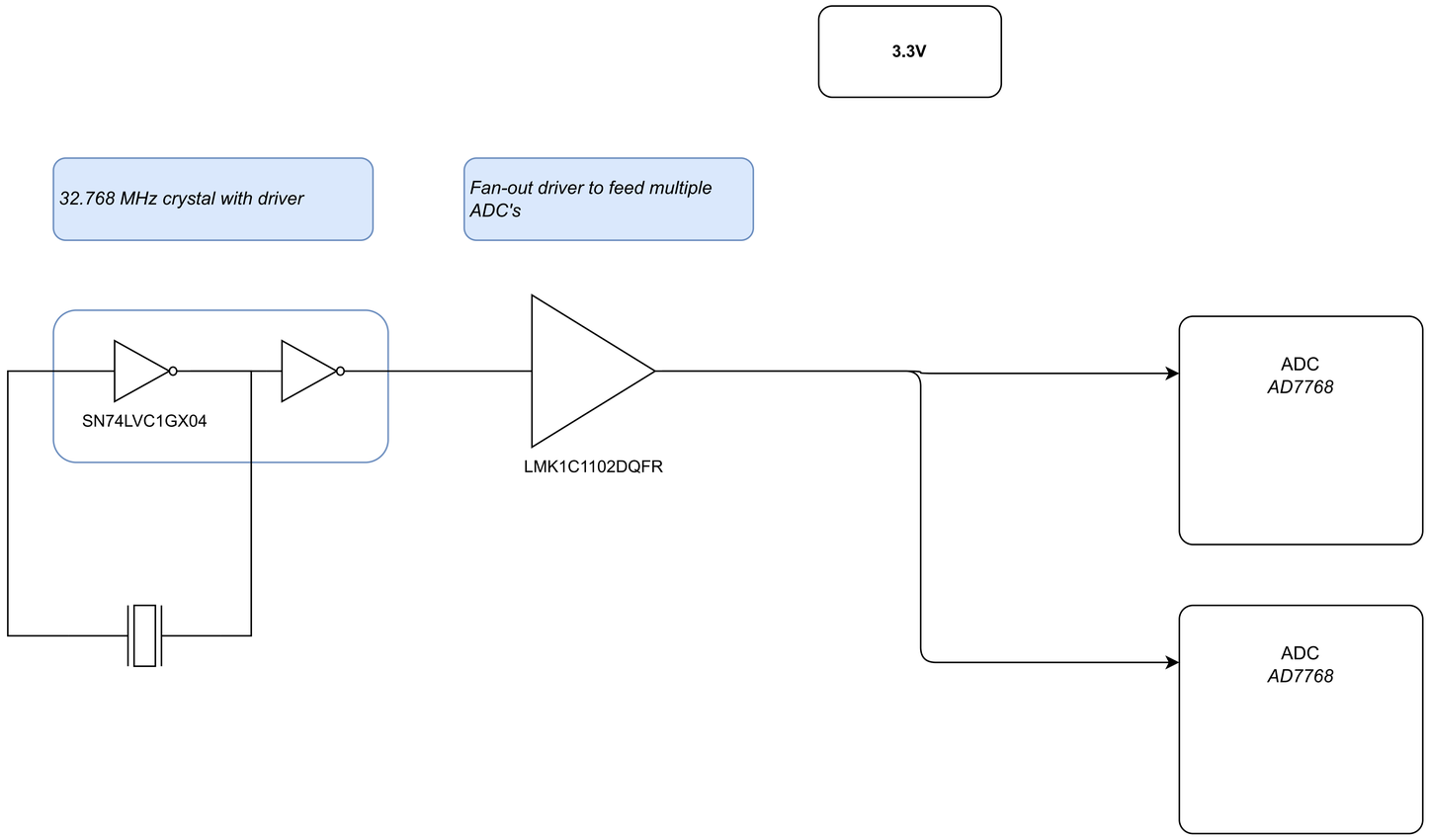
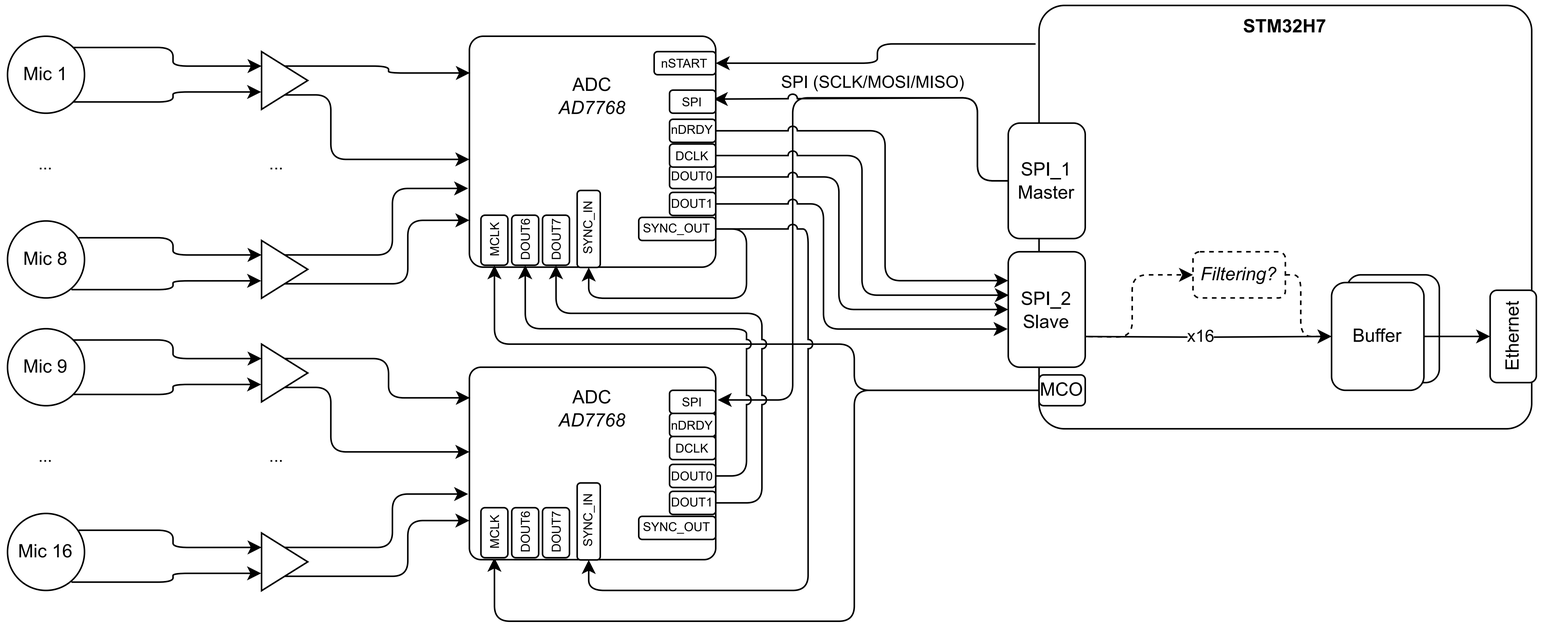

STM32H7
Mic 1
Mic 8
Mic 9
Mic 16
ADC
ADS1178
DR
SYNC
SCLK
MISO
MOSI
nDRDY
CLK
27MHz
MCO
SPI
Filtering?
x16
Buffer
Ethernet
A fCLK of 27MHz allows for a maximum data rate of 52kHz/channel
SPI SCLK and the ADC clock (fCLK) need to be of fSCLK/fCLK ratios of 1/1, 1/2, 1/4, or 1/8, etc

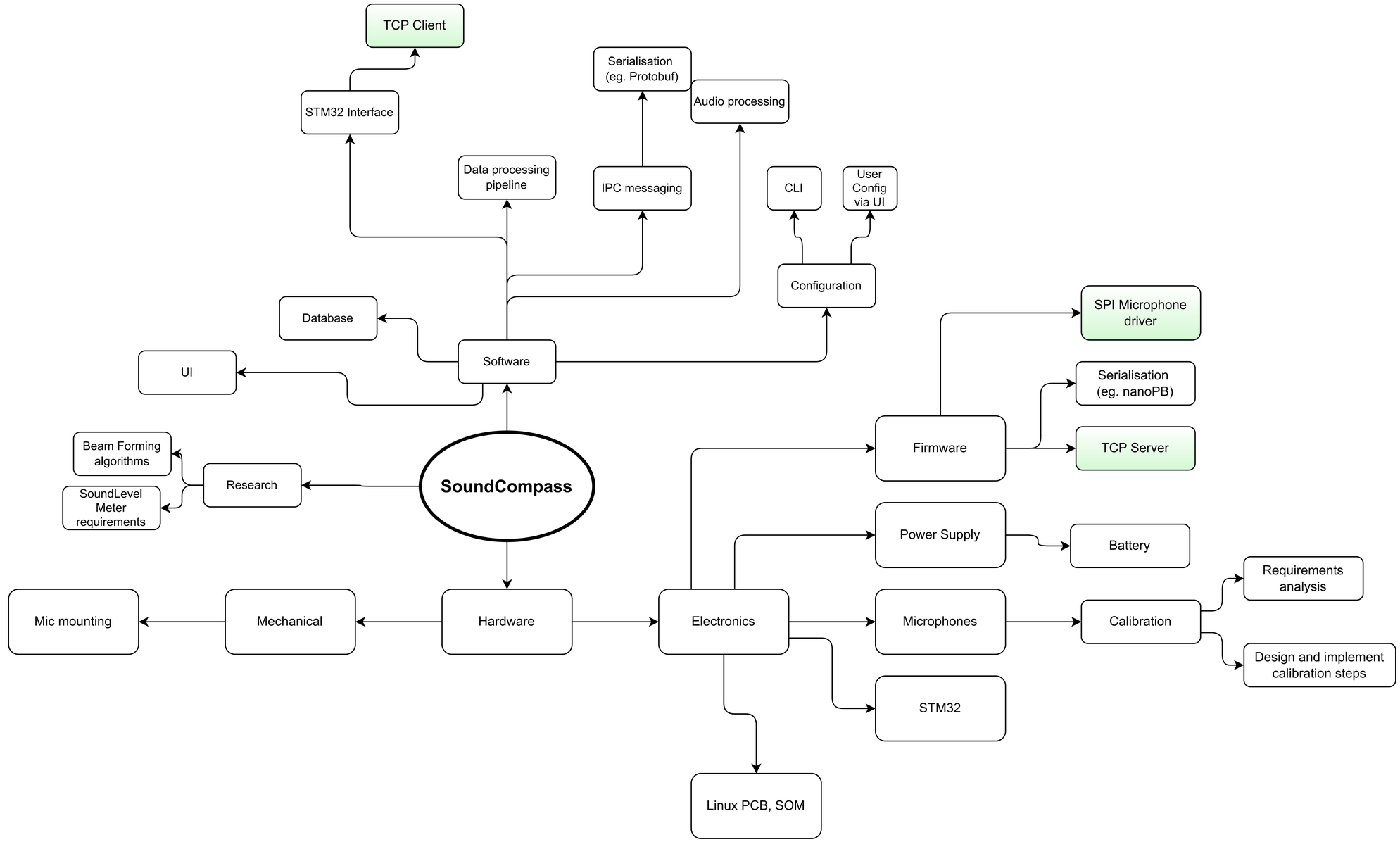

MEMS mics with differential output
Buffering, anti-aliasing
ADC's in daisy chained mode
nSYNC signal drives the sample rate and is attached to a timer
nDRDY goes low when a conversion result is ready to be read
STM32H7
Mic 1
Mic 8
Mic 9
Mic 16
ADC
AD7768
nSTART
SPI
nDRDY
DCLK
DOUT0
DOUT1
SYNC_OUT
SPI (SCLK/MOSI/MISO)
GPIO
SPI_1
Master
SPI_2
Slave
Active High
SN74HC1G04
Propagation delay matching buffers
Filtering?
x16
Buffer
Ethernet
GPIO/UART
32MHz oscillator
KC3225Z32.7680C1KX00
Fan out buffer:
LMK1C1102DQFR
The ADC's SYNC_IN signal must be synchronous to the MCLK. If this is not possible, there are 2 options to persue:
1. Send an asynchronous signal to nSTART, which will emit a synchronous signal out of SYNC_OUT
2. Use a SPI command to start the conversion, which will also emit a synchronous signal from SYNC_OUT
SPI_1 is the control and configuration channel
SPI_2 is a slave read-only channel for the ADC data.
When the ADC is operating in "One-shot" mode, the nDRDY line behaves differently to it's name: when data is ready to be ready out it goes high for the number of DCLK bits required to transfer the whole set of samples (from all the channels)
SPI_2 will be setup in dual-SPI mode to increase throughput while keeping the DCLK frequency low

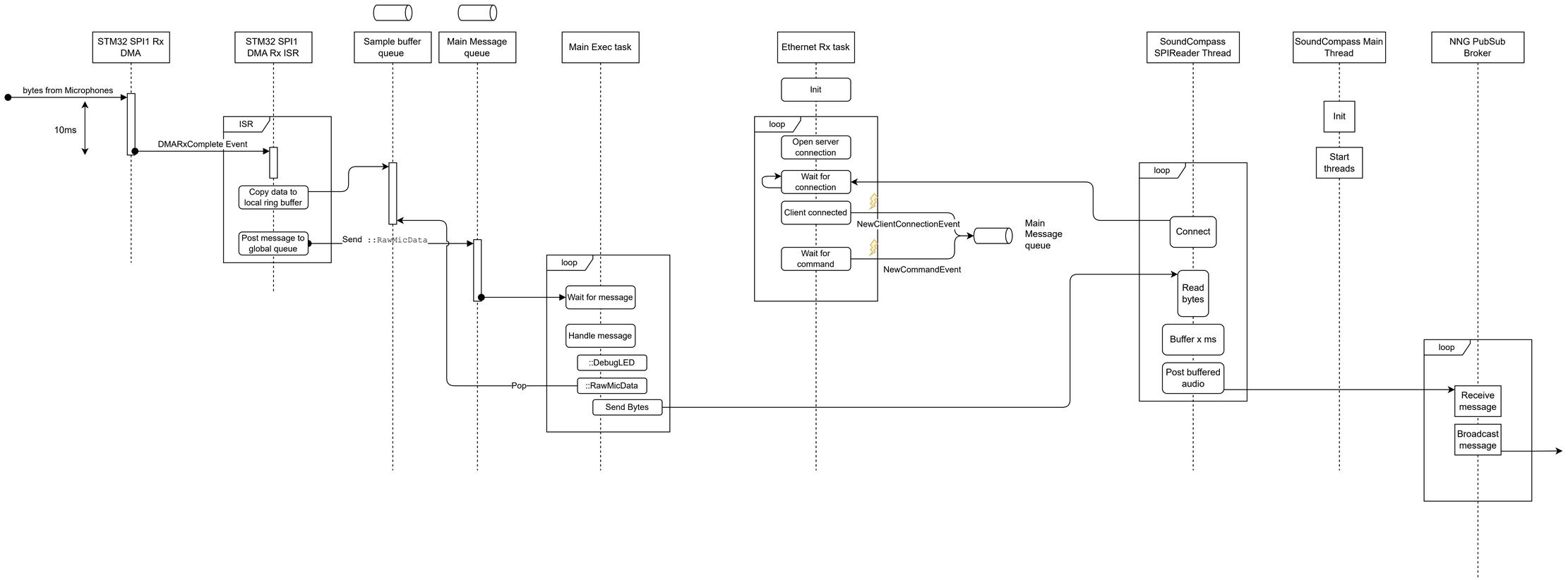

::IMessageBusClient
mic signals
RPC or req/rep socket
WebUI interface
processed results pulled from audioProcessor
onProcessedResult
pull socket
push socket
processed results pulled from audioProcessor
mic signal messages pushed to audioProcessor
QuadCore A53, Linux
- Main application
- rust
SoundCompass
IMicrophoneReader
channel
::SoundCompass
::IMessageBusClient
::IMessageBroker
- Reads audio from mics and pushes it to the main app
- main Sound Compass application
- passes mic signals to audioProcessor
- saves results to db
::ProcessingStage
::ProcessingStage
WebUI
- View data
- Control SoundCompass application
- View edit config
- Rust/Go/C#
Database
- System configuration
- Timeseries
- Candidates: TimescaleDB
UART/ GPIO
Ethernet
beamResult
audioFrame
audioProcessor
- Processing audio packets
- C++
main()
- parse CLI args
- create ::app
- call app.run()
audioProcessor::app
- create thread pool
- create ::audioProcessor
- create listener sockets
audioProcessor::beamProcessor
- entry point of the application
- parse CLI args
- start the app
- wait for exit signal
- listen for messages over the message bus (eg. NNG, ZMQ)
- push jobs to ::beamProcessor via a threadpool
- receive results from ::beamProcessor
- send result messages over the message bus
- responsible for processing the incoming mic signals passed in from the ::app
- performs the beam steering and 1/3 octave band filtering

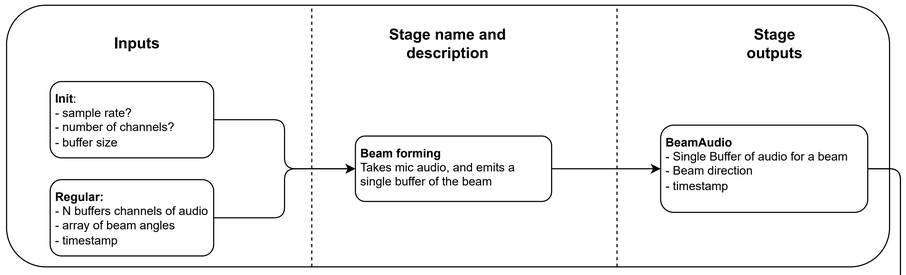

3.3V
32.768 MHz crystal with driver
Fan-out driver to feed multiple ADC's
SN74LVC1GX04
LMK1C1102DQFR
ADC
AD7768
ADC
AD7768

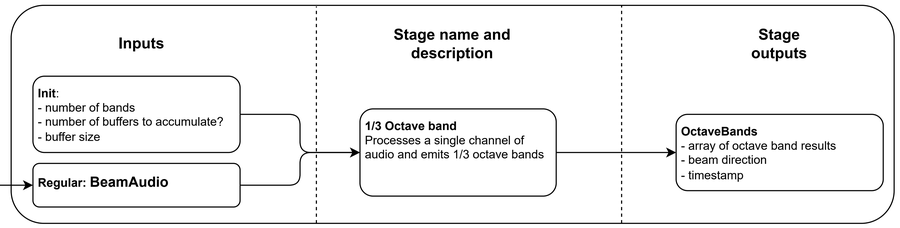


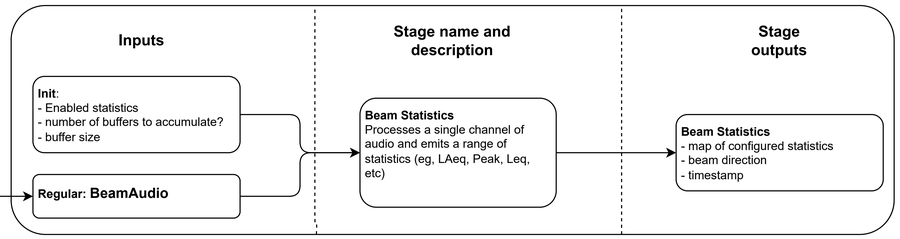
*The ADC's SYNC_IN signal **must** be synchronous to the MCLK. If this is not possible, there are 2 options to persue:*
1. Send an asynchronous signal to nSTART, which will emit a synchronous signal out of SYNC_OUT
2. Use a SPI command to start the conversion, which will also emit a synchronous signal from SYNC_OUT


SoundCompass
Research
Beam Forming algorithms
SoundLevel Meter requirements
Hardware
Mechanical
Mic mounting
Software
Database
UI
Data processing pipeline
STM32 Interface
TCP Client
IPC messaging
Serialisation (eg. Protobuf)
Audio processing
Configuration
CLI
User Config via UI
Electronics
Firmware
SPI Microphone driver
Serialisation (eg. nanoPB)
TCP Server
Power Supply
Battery
Microphones
Calibration
Requirements analysis
Design and implement calibration steps
STM32
Linux PCB, SOM


STM32 SPI1 Rx DMA
STM32 SPI1 DMA Rx ISR
Sample buffer queue
Main Message queue
Main Exec task
bytes from Microphones
10ms
ISR
DMARxComplete Event
Copy data to local ring buffer
Post message to global queue
Send ::RawMicData
loop
Wait for message
Handle message
::DebugLED
Pop
::RawMicData
Send Bytes
Ethernet Rx task
Init
loop
Open server connection
Wait for connection
Client connected
NewClientConnectionEvent
Main Message queue
Wait for command
NewCommandEvent
SoundCompass SPIReader Thread
loop
Connect
Read bytes
Buffer x ms
Post buffered audio
SoundCompass Main Thread
Init
Start threads
NNG PubSub Broker
loop
Receive message
Broadcast message





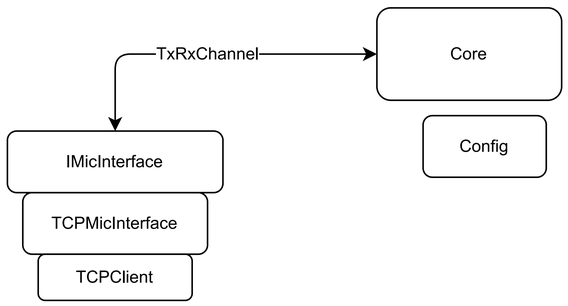


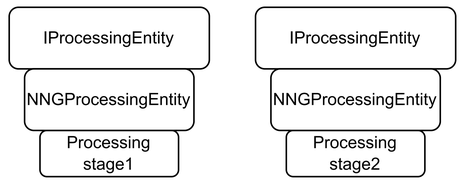
Messaging topology:
Instead of having a central PUB/SUB broker that broadcasts everything to everyone, each entity that can emit or listen for messages will be configured to listen ***only*** for the messages it cares about, and will broadcast to receivers who have specifically subscribed to the messages it emits.

The downsides are:
- quite a bit more configuraiton. Every entity will need to be configured with it's own identity, and the indentities of those to listen to.

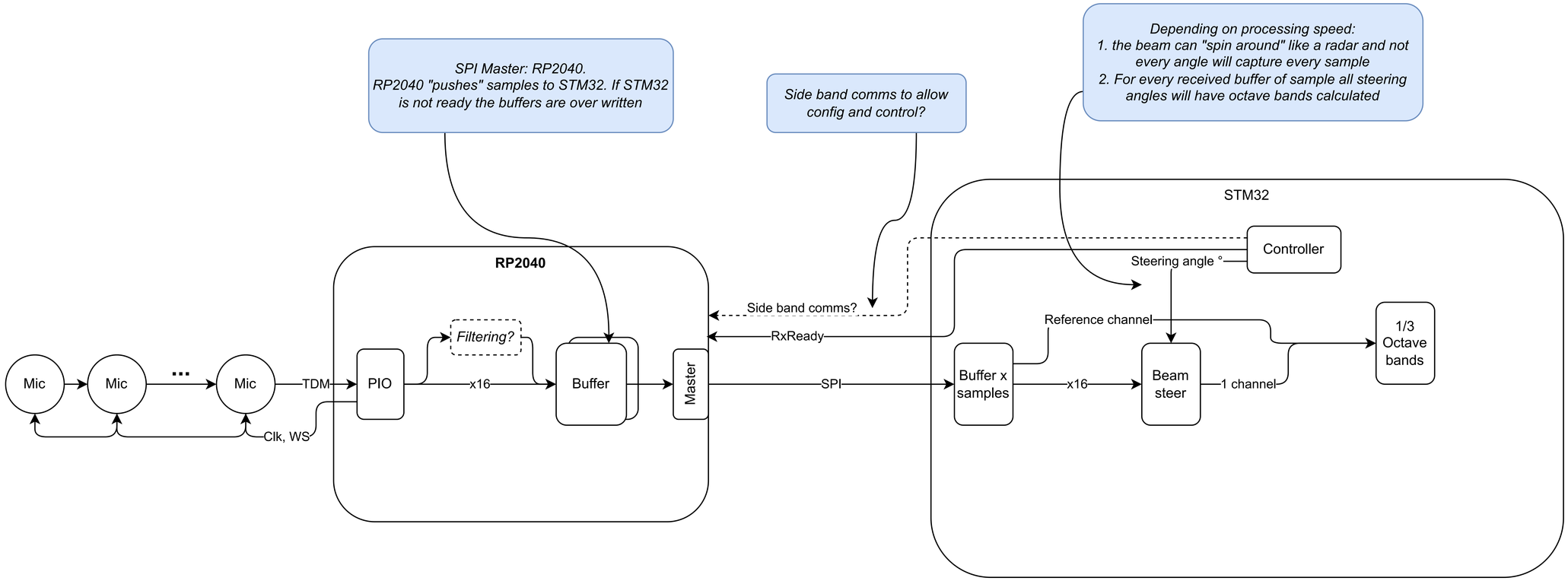
Stage Manager

- Creates stages
- Reads stage config
- creates named queues if they don't already exist
- passes a Rx and Tx channel to for the queues to the stage


TxRxChannel
Core
Config
IMicInterface
TCPMicInterface
TCPClient


IProcessingEntity
NNGProcessingEntity
Processing stage1

IProcessingEntity
NNGProcessingEntity
Processing stage2


SPI Master: RP2040.
RP2040 "pushes" samples to STM32. If STM32 is not ready the buffers are over written
Side band comms to allow config and control?
Depending on processing speed:
1. the beam can "spin around" like a radar and not every angle will capture every sample
2. For every received buffer of sample all steering angles will have octave bands calculated
STM32
RP2040
Controller
Steering angle °
Side band comms?
RxReady
Reference channel
1/3 Octave bands
Mic
Mic
...
Mic
TDM
Clk, WS
PIO
Filtering?
x16
Buffer
Master
SPI
Buffer x samples
x16
Beam steer
1 channel