

## Introduction to the Arduino Uno sketch

### Goal

Goal of the sketch is performing the QSK muting of your own delayed audio CW signal that is received via the WeSDR receiver.

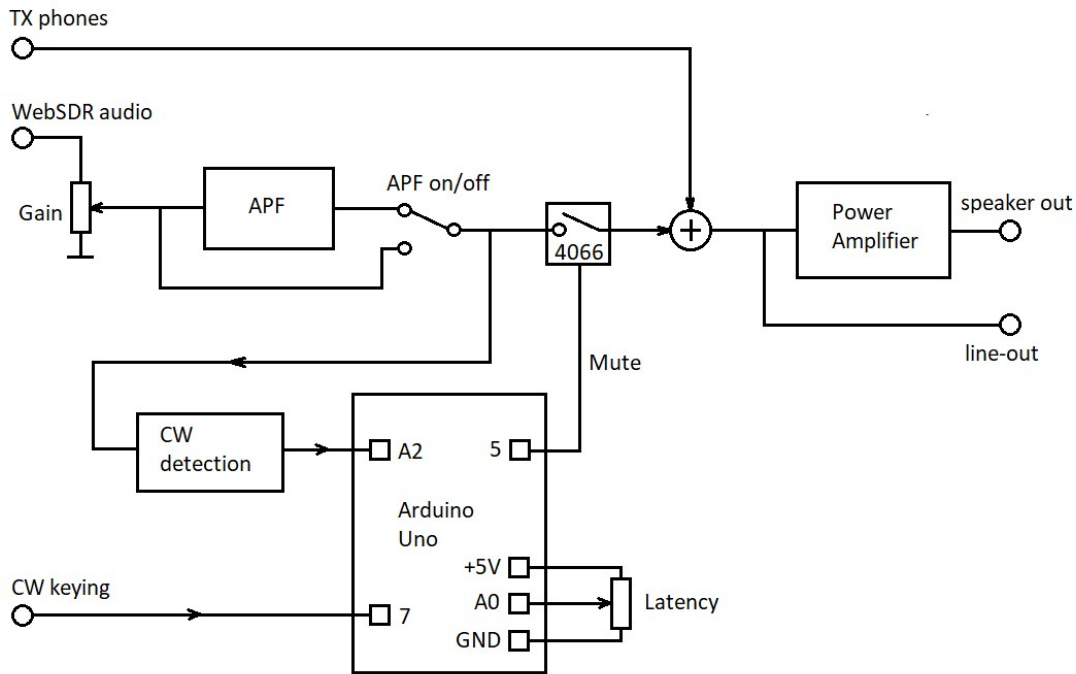


Fig.1. Block diagram of the hardware with the Arduino Uno.

### Automatic measurement of the latency

The delay in muting can be set manually by the potentiometer to the actual latency of the WebSDR. The latency however is not constant, has not a fixed value and can vary. It is much more convenient when muting tracks the current latency automatically.

The program therefor has to measure the time difference between the transmitted and received dots and dashes. A valid measured time difference has to be between the expected minimum (**latency\_min**) and maximum latency (**latency\_max**).

Only when transmitting CW (**transmitting**) the latency can be measured. This period starts at the first dot or dash and ends after the last transmitted dot or dash extended with the maximum expected latency time.

Only after that time we know for sure that all transmitted dots and dashes are received.

## CW detection

The CW detection circuit is a simple envelope detector.

The AGC in the WebSDR tries to keep the amplitude constant. That is a disadvantage for this method of CW detection. It can be used for stronger CW signals. However it needs a more complex processing in the Arduino Uno to compensate.

The Arduino Uno samples the amplitude at a 500Hz frequency.

Only during transmitting the peak amplitude (**peakAMP**) is tracked with a different attack and decay speed. The received CW detection (**myCW**) threshold is set at 75% of the peak amplitude. This level depends e.g. on the AGC behavior of the WebSDR.

Prior to the latency calculations a spike filtering is performed. Dots and dashes shorter than 10msec (**spikeT**) are considered a spike and are discarded.

The peak amplitude is not tracked when we don't transmit (**transmitting**). Detection is still performed, but not used except for driving the LED.

The analog envelope detector and this processing introduce a delay which has to be corrected for in the Arduino Uno sketch (**det\_delay**).

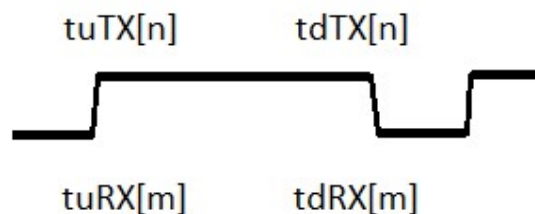
The analog amplitude signal of the envelope detector has an offset voltage. In the sketch the correction (**AMP\_offset**) has to be set to the actual offset voltage.

## Selection manual or automatic

By setting the manual latency potentiometer to the lowest setting the automatic calculated latency is selected and used.

## Latency calculations

The successive transmitted and received dots and dashes start and end times are stored in circular buffers with a length of 10 (**tuTX/tdTX** en **tuRX/tdRX**).



At any time the start and end times of the transmitted (TX) and the delayed received CW signal (RX) last 10 dots or dashes are present in the buffers. Of each dot and dash the average time is then calculated. This makes two rows of 10 average times (**taTX** and **taRX**). Averages are only valid if dots and dashes are completely passed.

The latency calculation routine is run directly after a full dot or dash is received and only during the CW transmit period (**transmitting**).

The last 4 dots or dashes of the received CW signal are used for the measurement of the current latency. The pattern in time of the 4 corresponding average times (**taRx/taRXc**) is compared with 4 successive times in the 10 average times of the transmitted CW (**taTX/taTXc**).

First both 4 average times are subtracted. The 4 differences (**diff**) between the average times are equal if the pattern in time of the dots and dashes matches. If the pattern in time doesn't match, the 4 differences are not equal and differ widely. Calculated is the sum (**sumres**) of the absolute value of the difference of the 4 differences (**diff**) with the average difference (**mean\_diff**).

In reality there is always noise on the measured times. The pattern in time of dots and dashes matches if the sum (**sumres**) is below a threshold (**lat\_thres**).

Only if this happens ones (**match\_count**) over the 10 average times of the transmitted CW (**taTXc**) the match is unambiguously and the measured time difference is accepted as the current latency.

Unambiguously is not possible when e.g. only dots are transmitted.

So the latency is not always updated after every received dot or dash.

This threshold is an estimate of the expected noise on the timing of the received detected CW signal. It also minimizes the change that breaking-in CW signals are seen as own CW and causing a wrong latency.

## Variations in de latency

The latency of the WebSDR exhibits short term variations which cannot always be measured in time. Manual setting of the latency will always deviate from the current latency.

These small variations in latency are mitigated by an extra pre and post muting time. Muting starts earlier and stops later over a time set by **var\_latency**.

## Mute signal for the 4066 switch

The mute signal (**Mute**) is calculated from the content of the two circular buffers containing the start and end times of the transmitted CW (**tuTX** en **tdTX**). When the current time is higher than a start time (**tuTX**) plus the latency and smaller than the corresponding end time (**tdTX**) plus the latency, the audio is muted.

## CONT and OFF

The three-way switch enables a selection for muting all audio (**OFF**) or no muting (**CONT**). No muting enables hearing the our own delayed CW.

## Main loop frequency 500Hz

In order to keep the analog CW detection predictable the main loop time is kept at 2msec by introducing an extra delay.

## Adjusting the det\_delay

The analog envelope detector and the detection process in the Arduino Uno introduce a delay in the received CW. In the sketch **det\_delay** corrects for this extra delay.

Adjusting **det\_delay** can be done by ear. More accurate is measuring the latency in parallel using an oscilloscope with on one channel the transmitted RF signal and on another channel the received WebSDR audio.

## RF detection

The CW keying signal is provided by most transceivers and external keyers. An alternative method is detecting the RF energy on the coaxial cable to the antenna. Then only when transmitting CW the WebSDR audio is muted. In the same way it can be used for SSB.

See: RF\_detection\_circuit.png

## Is a length of 10 adequate for the circular buffering?

Suppose we use the last 4 received dots/dashes for the latency measurement.

These 4 have still to be in the TX buffer after the worst case latency.

During this worst case latency period new dots/dashes can be transmitted.

6 locations in the TX buffer remain for these new dots/dashes.

The number of new dots/dashes is largest for worst case latency, for dots and for the highest CW speed.

12wpm are 5 dots/sec

30wpm are 12.5dots/sec and 6.25 dots per latency\_max 0.6sec.

=> for 30wpm a length of 10 is suffice

At higher CW speeds however it will not directly go wrong.

Practically never 6 dots are sent directly after each other and latency is normally always <0.5sec.

## Algorithm development

The algorithm for automatically tracking the current latency in the audio of the WebSDR first is first developed in Matlab. See: QSK\_CW\_Arduino.m

The free available Octave can also be used:

<https://www.gnu.org/software/octave/>

The algorithm is developed with processing on the Arduino Uno in mind. The processing shouldn't need to much processing time or be to complex. During testing the Matlab/Octave implementation is used and modified regularly.

Important is reliability. A small deviating latency is preferred over a completely wrong latency. That is why a higher acceptance threshold (lat\_thres lower) for the update with the measured latency is preferred.

Not everyone runs Matlab or Octave. That is why an Excel version is available with the principle of automatic tracking the latency (QSK\_CW\_arduino.xlsx).