DIGITAL SIGNAL PROCESSING FOR COMMUNICATIONS AND RADAR LAB

Reinhard Feger, Lukas Furtmüller

Summer Semester 2024

Institute for Communications Engineering and RF-Systems

JOHANNES KEPLER UNIVERSITY LINZ

Today's Contents

Additional Debugging via UART

- The development board offers a UART connection over the same USB also used for debugging
- Over this connection data can be sent to the PC, where it is easier to analyze (for example graphically)
- The data rate of 115200 bps is sometimes not sufficient for continuous data streaming, but blocks of data can be investigated together with the debugging pause/step functions
- Check the Device Manager to find out the number of the com port used for the UART$\leftrightarrow$USB interface
- Tools on the PC: SerialPlot[1] and HTerm[2]

[1] https://github.com/hyOzd/serialplot

[2] http://www.der-hammer.info/pages/terminal.html

Additional Debugging via UART

- Most basic commands to transmit/receive data to/from the PC (for HAL):
  `HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)` and
  `HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)`
- To wait forever, `HAL_MAX_DELAY` can be used for the timeout parameter
- The `Size` parameter defines the number of bytes
- The handle `huart4` needs to be used to access the UART connection over the USB debugging interface

Exercise 1: Audio pass through and plotting

- Exercise 1: Extend your program to read audio input and replay the read signal at the output without modifications
- Use the functions

```
void wm8731_waitInBuf(struct wm8731_dev_s *self)
```

```
void wm8731_getInBuf(struct wm8731_dev_s *self, int16_t *data)
```

  and define your own own `adc_buffer` array with 256 samples (128 even samples left, 128 odd samples right)
- Start the audio ADC in a similar way as it is done for the DAC in the template
- Verify that the pass through works by listening to an audio signal
- Plot (pieces of) the audio signal on the PC using the UART interface. What happens when you listen to the audio in this case?

Block Processing

- As can be seen from the previous examples, the continuous data stream is divided into blocks
- Consequently also processing needs to be implemented on a per block basis
- Widely used methods are
  - Overlap add and
  - Overlap save
- Especially in combination with the fast-Fourier transform efficient real-time processing of continuous signals can be implemented

Overlap Add

- Assume a long (infinite) signal $x[n]$ that should be filtered by a filter with a short (length M) impulse response $h[n]$
- The convolution can be written as

$$y[n] = \sum_{m=-\infty}^{\infty} h[m]\, x[n-m] \tag{1}$$

- Since h vanishes outside its length M, (1) can be simplified to

$$y[n] = \sum_{m=0}^{M-1} h[m]\, x[n-m] \tag{2}$$

Overlap Add

- Applying (2) to blocks of length L which are taken from $x[n]$ according to

$$x_k[n] = (\sigma[n] - \sigma[n - kL])\, x[n] \tag{3}$$

  with the discrete time unit step

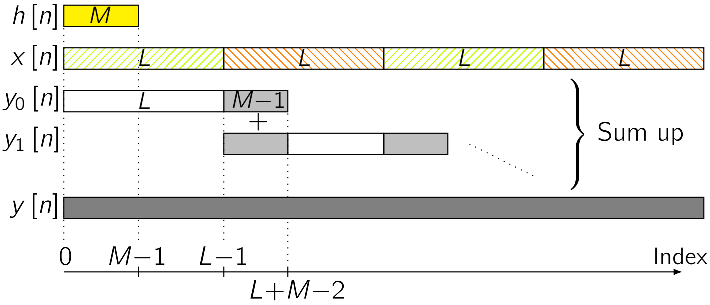
$$\sigma[n] = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$$

  leads to convolution segments of length $L + (M - 1)$.
- Using (3) allows to write the convolution result (2) as

$$y[n] = \left(\sum_k x_k\right) * h = \sum_k (x_k * h) = \sum_k y_k \tag{4}$$

Overlap Add

- It can be seen, that in (4) segments of length $L + (M - 1)$ are summed up, but they are only shifted by L samples. According to (4) the overlapping samples of neighboring regions need to be added
- Graphically this can be represented as follows:



Exercise 2: FIR Filter

- Implement a digital low-pass filter using the overlap add method with the following filter parameters
  - Sampling rate: $8\,$kHz
  - Corner frequency: $1\,$kHz
  - Order: 63
  - Window: Hamming
  - Block size: 128
- Calculate the filter weights using the window method in Python (`scipy.signal.firwin`) or Matlab (`fir1`)
- Prepare a Python or Matlab implementation of the overlap add method, to simplify debugging

Exercise 2: FIR Filter

- Method 1: Standard FIR implementation

$$y[n] = \sum_{m=0}^{M-1} h[m]\, x[n-m]$$

  with $N \geq L + M - 1 \Leftrightarrow N - M + 1 \geq L$
- To improve efficiency, L should be as large as possible
- Use a buffer storing $M - 1$ output samples from one block to add them to the first output samples of the next block
- Tips for testing (for example in combination with SerialPlot):
  - Use an impulse response consisting of a single Dirac. The signal should be simply passed through in this case
  - Use a single Dirac as input signal. The output should be the impulse response of the FIR filter
  - Use a simulated continuous ramp signal to check for non-desired steps at block transitions

Exercise 2: FIR Filter

- Method 2: Realize an improved implementation reducing the number of inner loop runs

$$y[n] = \sum_{m=0}^{N/2-1} \left(h[m]\, x[n-m] + h[m+N/2]\, x[n-m-N/2] \right)$$

- Method 3: Use optimized DSP functions from the Cortex Microcontroller Software Interface Standard (CMSIS) DSP Software Library. See:

  https://arm-software.github.io/CMSIS_5/DSP/html/group__FIR.html
- The processor offers a floating point unit, relevant functions are `arm_fir_init_f32` and `arm_fir_f32`
  - Optionally you can also use the corresponding fixed point counterparts
- Test which realizations (manually implemented or using library functions) are fast enough to support a sampling rate of 48 kHz