Educational platform for laboratory works in Digital Signal Processing

Authors:

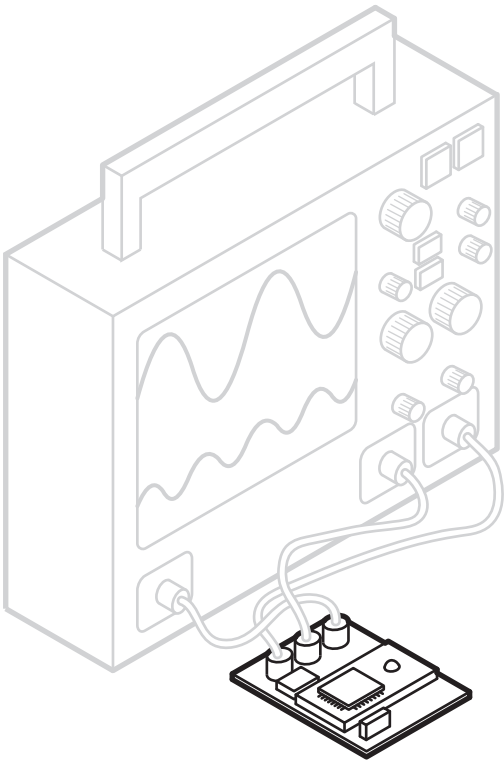
Ilia Kempi
Yani Mustapha

Supervisor:

Thierry Baills
Senior lecturer

Metropolia University of Applied Sciences
Helsinki, Finland

February 25, 2015



Abstract

Teaching digital signal processing often involves comparison between discrete and continuous linear time-invariant systems. Numerical data processing is usually demonstrated by simulation or hardware implementation of a particular algorithm. An inexpensive educational platform, capable of emulating fundamental digital filters has been developed and integrated into teaching process.

The hardware module is based on advanced microcontroller supported by signal conditioning circuitry. System is configured from PC application with simple user interface that allows to easily model, test and evaluate various digital filter configurations. Finite and infinite impulse response kernels can be implemented in real time with sampling frequency up to 100 kHz.

Contents

Acronyms

ADC Analog-to-Digital Converter. 6, 9, 13, 15, 30

ASIC Application-specific Integrated Circuit. 24, 30

CMSIS Cortex Microcontroller Software Interface Standard. 7–9

DAC Digital-to-Analog Converter. 6, 9, 14, 15, 30

DSP Digital Signal Processing. 5, 7–9, 11, 14, 15, 27, 29, 30

EMC Electromagnetic Compatibility. 27

FFT Fast Fourier Transform. 7, 19, 23

FIR Finite Impulse Response. 5, 10–14, 20

GUI Graphical User Interface. 6, 7, 9, 15, 23

IIR Infinite Impulse Response. 5, 10–13, 20

ISR Interrupt Service Routine. 13, 14

PCB Printed Circuit Board. 23–25

RPC Remote Procedure Call. 7–9, 12, 13, 23

RTC Real-Time Computing. 5, 9, 13, 15, 20, 29

1 Introduction

For several years, basics of Digital Signal Processing (DSP) have been taught in both Information Technology and Electronics departments of Metropolia University of Applied Sciences. As this field demands complex background of calculus, computer science and processors, it is very important to understand the basics in order to advance.

Purpose of this project is to develop a simple programmable platform, capable of basic DSP operations. Users should be able to configure the DSP-related properties of the device without interfering with low-level implementation layer. Ideally, student would test recently learned concepts on the hardware, thus convincing themselves in the applicability of the fresh knowledge.

Fundamental topics, studied in the beginning of DSP theory are Sampling, Z-transform and Discrete Convolution. Students are introduced to quantization, mirror band, impulses, filter kernels and their implementations. With this in mind, desired features of the system are following:

- Customizable Finite Impulse Response (FIR) filter functionality
- Customizable Infinite Impulse Response (IIR) filter functionality
- Processing of numerical input sequences in digital format
- Real-Time Computing (RTC) of continuous signals with sampling and reconstruction
- User input can not damage the system

This report contains technical information about implementation of the device. Thorough explanation of system architecture and reasoning behind the decisions can provide insights and background for future improvements.

2 Implementation plan

2.1 Front and back end system

The idea of implementing the digital signal processing on the actual hardware is demonstrated at best with Field-Programmable Gate Array or a DSP processor. However, in comparison of cost and complexity of the system to the basic features that it needs to provide, DSP processor is an overkill. To run a simple filter kernel, an average multi-purpose microcontroller can

perform as good as specialized device. Moreover, the system should have customizable processing core, therefore its parameters must be flexible for experimenting purposes.

To provide a rich and friendly user interface for the students, PC application is obviously the best solution. Graphical User Interface (GUI) can play the role of isolated front end panel, while all processing activity is wrapped inside the hardware (back end). Following this logic, best way of realizing the system is to use a client-server approach, illustrated in the figure 1.

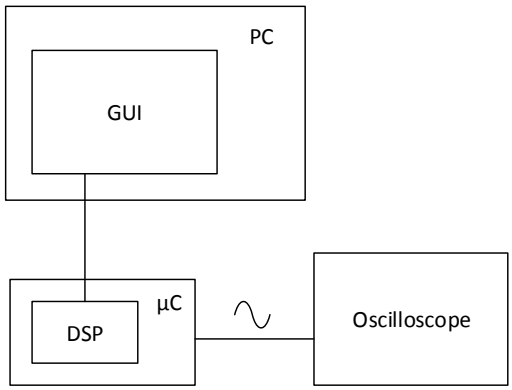


Figure 1: Generic system structure

Typically, instructions and data are exchanged between front and back ends via serial interface, which is easily realizable with almost any modern microcontroller.

2.2 Hardware and software

Back end device is based on the popular MBED development board, which features powerful NXP LPC1768 ARM CORTEX-M3 based microcontroller. Main characteristics of interest are:

- 32-bit processor running on 96 MHz clock
- 12-bit Analog-to-Digital Converter (ADC) with up to 200 kHz sampling rate
- 12-bit Digital-to-Analog Converter (DAC)
- Voltage regulator to operate only from USB supply

Because of MBED modular nature, the development board could be easily reusable for other projects, when the device is not needed. There are many useful libraries, developed for Cortex Microcontroller Software Interface Standard (CMSIS) and the large ARM MBED developer community, both are positive factors for rapid development of the system.

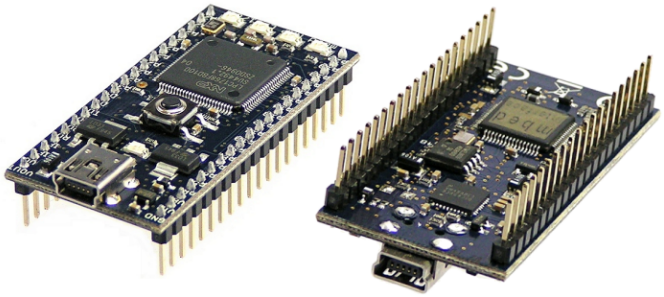


Figure 2: MBED development board with NXP LPC1768

The microcontroller board is easily programmed via USB interface. Precomplied binary files can be transferred on the internal flash memory of the system with any file manager. This allows to program or update the device even without specialized software.

To assist the development, MBED community offers a simple on-line environment with browser interface, project manager, code editor and compiler. While it is still possible to use powerful software tools such as KEIL or IAR, the back end program was developed in the MBED online environment.

Front end user interface is an application based on MATLAB GUI. MATLAB was selected as runtime environment for following features:

- Modeling and simulation of DSP algorithms
- Powerful data processing features
- Extensive programming and GUI capabilities
- Compatibility with numerous devices

MATLAB computing environment offers many mathematical functions such as Fast Fourier Transform (FFT) and allows to create even more complex data processing routines based on its own programming paradigm. Program implemented on this environment is limited in terms of portability, requiring MATLAB instance running on the PC. On the other hand, proof of complex mathematical concept can be developed relatively fast.

Communication between front and back end is realized with Remote Procedure Call (RPC) via Serial COM port. With this interface, GUI application

can execute methods on the microcontroller remotely, while necessary information is passed either as a function parameter or as a return value.

Before studying about the educational platform from this report, it is strongly recommended to get familiar with its capabilities by reading the user manual. The latter is supplied together with the report as a separate **Digital Filter Workbench manual.pdf**. Source codes of the back end and front end programs are embedded into this report PDF document as **dsp-fw.cpp** and **testbench.m** accordingly.[1] Program structure and logic are described in the following sections, including alphabetic variable reference for each source file.

3 Back end application

3.1 Program structure

Microcontroller program is designed around two most important libraries: MBED-DSP, general DSP library for CORTEX microcontrollers under CMSIS [3] and MBED-RPC, which allows external client program to execute firmware functions via RPC [4]. Block diagram of the program structure is shown in figure 3.

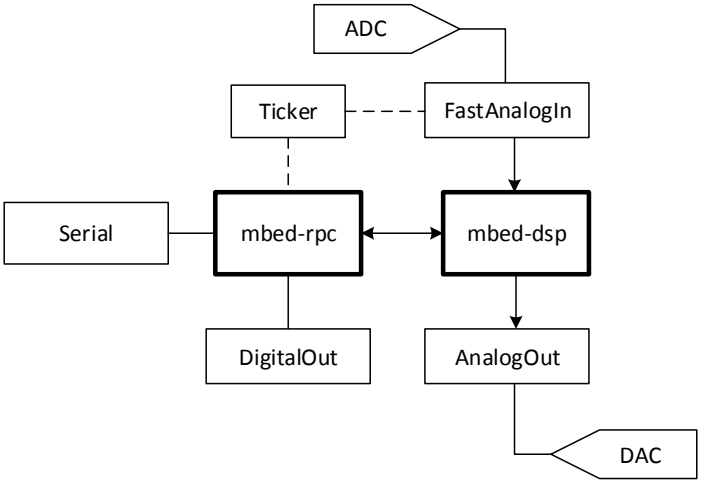


Figure 3: Microcontroller program structure

[1] In order to access file attachments of the PDF document, one should use an advanced PDF reader, for example ADOBE READER or FOXIT READER.

In standby mode, SERIAL object monitors the serial port, passing all the data received from PC into RPC library methods. After it fetches the instructions, two scenarios of data processing can be invoked:

Numerical mode: Ready sampled signal is received from the client program in packets. Processed packets are sent back to the GUI.

RTC mode: Continuous signal from a function generator is processed by sampling it with inbuilt ADC and result is forwarded to the DAC.

In numerical mode, a custom routine passes incoming data into local buffer and executes DSP library functions on this array. Processed data is returned back to PC from output buffer in similar way.

If user instructs the device to process a continuous signal, DSP functions are redirected to listen for FAST ANALOG IN object and output into ANALOG OUT. TICKER timer is set to control the sampling rate and DIGITAL OUT is used to indicate that system is going into RTC mode. After that, program listens only to ADC, and unable to receive any commands from the client program until microcontroller is reset.

Fixed point arithmetics is used in the program to increase RTC performance. Compared to the floating point, hardly delivering 30 kHz processing rate, 15-bit fixed-point arithmetics execution can exceed 100 kHz, giving significantly larger bandwidth for DSP experiments. Numerical mode is using same fixed-point functions for consistency of the results between two modes. Approximate signal to quantization noise ratio then would be:

$$SQNR_{q15} \approx n_{bits} \cdot 6[dB] = 80[dB] \tag{1}$$

3.2 Signal processing functions

Digital filter functions, provided by CMSIS DSP library [3] are set to work with simple initialization procedure. First, data structures FIR and IIR are created for each type of filter accordingly. Functions of type INIT are utilized to assign the environment variables into the structure, so that the processing function has concrete filter instance (kernel, delay line history) to work with. At processing function call, input and output buffer pointers BUFFERIN[ ] and BUFFEROUT[ ] are passed in as arguments, together with variable specifying block size. Latter value tells how many points processing function should take from the buffer.

FIR filter

This filter algorithm is based upon a sequence of multiply-accumulate operations. Each filter coefficient is multiplied with a delayed input, such that:

$$y[n] = \sum_{i=0}^{c-1} x[n-i]b[i] \tag{2}$$

where b directly corresponds to FIR_COEFFS[ ] and c is a number of filter coefficients in the array, which is stored in the NUM_TAPS variable. Algorithm block diagram is presented in figure 4.

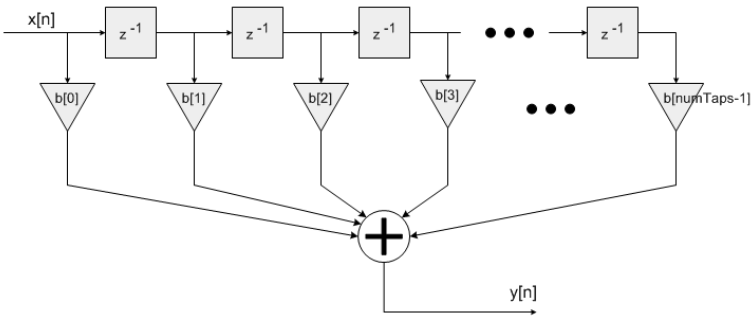


Figure 4: FIR block diagram

Block size of data to be processed in one FIR function call is specified in BLOCK_SIZE. For processing blocks smaller, than the length of the kernel, previous signal samples are stored in FIR_STATE[ ].

Biquadratic IIR filter

This is a common second-order recursive filter algorithm, which consists of three feed-forward and two feed-back coefficients, as shown in figure 5.

Direct form filter implementation is preferred instead of transposed form mostly because it is straightforward and easy for beginners to start with. However, if taking closer look to the figure and its transfer function (equation 3), one could notice that denominator polynomial coefficients have a negative sign, being different from a canonical $H(z)$ form.

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 - a_1 z^{-1} - a_2 z^{-2}} \tag{3}$$

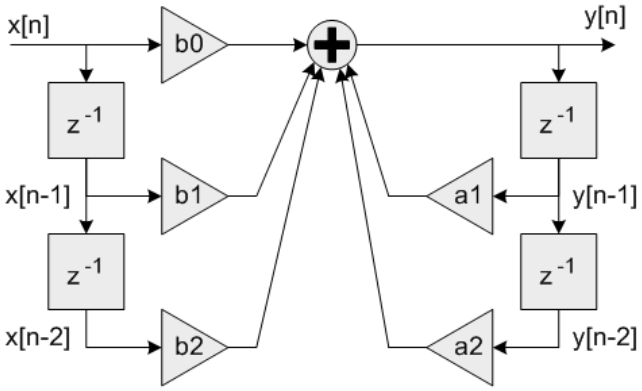


Figure 5: IIR block diagram

Multiple stages of the filter can be set up, so that output of first stage becomes input of the second stage, and so on. Total output of the process would be:

$$Y(z) = X(z) \prod_{n=1}^{s} H_n(z) \tag{4}$$

where s is a total number of second-order stages, specified in NUM_TAPS2. Transfer function coefficients are stored in the IIR_COEFFS[ ] in special order. If some filter coefficient should exceed one, whole array can be scaled exponentially with factor $-2n$, and n is written to PSHIFT variable.

Similarly to FIR algorithm, block size of data to be processed with IIR function call is specified in BLOCK_SIZE2 and previous signal samples are stored in IIR_STATE[ ].

3.3 Remote procedure call functions

The library for MATLAB communication allows to use hardware abstraction objects, call functions and access single variables on the MBED program through serial port. Because DSP involves exchanging huge amounts of data, structures similar to shared arrays should be implemented. However, limitation on the number of serial objects makes accessing single variables inefficient for this purpose, compared to using remote functions.

Buffers for data processing are defined as global arrays which can be accessed with any internal method, so that processing and transporting functions that are called from MATLAB can govern the internal state of the program.

In order to make function RPC compatible, its arguments should be of special data type:

```
func(Arguments* zzz, Reply* yyy)
```

To make function available to the MATLAB, RPC FUNCTION object should be created. Then, the new method will be seen by the specified label.

```
RPCFunction function(&func, "func-label");
```

To access the arguments sent from the client program with the function call, GET ARG method is used to retrieve them consecutively. Array of return values is constructed from subsequent arguments of PUT DATA function.

```
zzz->getArg<arg_type>();
```

```
yyy->putData(arg);
```

Each function operates in its own scope. First incoming argument will usually tell the function in which state it must operate, e.g. should it read or write data from the buffer. Functions and their sub-states are listed here:

buffering

1. Receives one sample from client program and its index in the block. Converts the sample into Q15 fixed-point format and writes it into BUFFERIN[ ] array on the index position.
2. Gets one sample from the BUFFEROUT[ ] array by the index received from the client program. Converts the sample to floating point format and returns the value.

coeffs

1. Receives one FIR coefficient value from client program and its index in the kernel. Converts the value into Q15 fixed-point format and writes it into FIR_COEFFS[ ] array on the index position.
2. Receives BLOCK_SIZE and NUM_TAPS as two consecutive integer arguments.
3. Receives one IIR coefficient value from client program and its index in the kernel. Converts the value into Q15 fixed-point format and writes it into IIR_COEFFS[ ] array on the index position.
4. Receives BLOCK_SIZE2 and NUM_TAPS2 as two consecutive integer arguments.

firfilter

1. Initializes the FIR filter instance with the given state variables NUM_TAPS, FIR_COEFFS[ ], FIR_STATE[ ] and BLOCK_SIZE.
2. Processes BLOCK_SIZE samples from BUFFERIN[ ] and writes result to BUFFEROUT[ ].

iirfilter

1. Negates the denominator polynomial coefficients (as explained on page 10). Initializes the IIR filter instance with the given state variables: NUM_TAPS2, IIR_COEFFS[ ], IIR_STATE[ ] and BLOCK_SIZE2.
2. Processes BLOCK_SIZE2 samples from BUFFERIN[ ] and writes result to BUFFEROUT[ ].

launcher

1. Receives S_PERIOD as an integer parameter. Flags IS_RT and FIR_RT are set to true, initiating the FIR RTC mode.
2. Receives S_PERIOD as an integer parameter. FIR_RT flag is set to false and IS_RT to true, initiating the IIR RTC mode.

3.4 Real-time operation

Slow serial communication makes it impossible to simultaneously show real-time sampling results to the front end application and vice versa. Because of that, such kind of demonstration is avoided completely, and full RTC mode is introduced. In this procedure, device is reconfigured to ignore communication and constantly process the data. Operation states are presented in figure 6.

Standby mode corresponds to constant listening of serial port and RPC command parsing. In this state, the device is able to receive instructions and process data sent from the front end. Every loop iteration, program quickly checks IS_RT flag and resumes operation.

RTC mode is initiated with call to LAUNCHER function, mentioned in the previous section. It changes the IS_RT and forces the program to exit the standby mode. Then, FIR_RT flag is checked and either filter processing mode is set up with INIT type of function. Special Interrupt Service Routine (ISR) is attached to the SAMPLER timer in such way that it will be executed with microsecond intervals specified by S_PERIOD integer.

System locks in an infinite loop. In order to avoid long floating-point to fixed-point conversion, ADC sample is read from ANIN object directly to 16-bit integer variable INN. The value is fed into processing function with data size

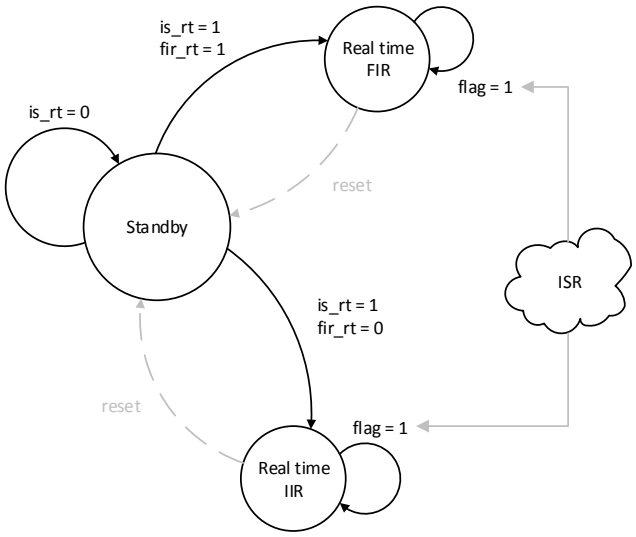


Figure 6: Device operation states

argument being one. Fixed-point result is put into OUTT and then implicitly cast into integer for the DAC interface ANOUT.

Loop operations take less than 10 microseconds to execute, which is experimentally determined. The process is harmonized with boolean FLAG. It inhibits iteration of the loop, until SAMPLER timer ISR flips the value back. To recover into communication mode, user must hard-reset the microcontroller with the dedicated button on the board.

dsp-fw.cpp global variable reference

anin (FastAnalogIn object) Hardware abstraction for ADC. 13

anout (AnalogOut object) Hardware abstraction for DAC. 14

block_size (unsigned int 32) Size of packet to be processed with FIR filter. 10, 12, 13

block_size2 (unsigned int 32) Size of packet to be processed with IIR filter. 11–13

bufferin[ ] (Q15 fixed-point) Buffer for storing incoming blocks with signal data prior to processing. 9, 12, 13

bufferout[ ] (Q15 fixed-point) Buffer for storing outgoing blocks with processed signal data. 9, 12, 13

fir (struct) Data instance of FIR filter which stores operating environment for the processing function. 9, 13

fir_coeffs[ ] (Q15 fixed-point) Stores FIR filter kernel in time-reversed order. 10, 12, 13, 20

fir_rt (boolean) Flag which controls whether program enters FIR or IIR RTC loop. 13

fir_state[ ] (Q15 fixed-point) Stores temporary FIR variables, array is constantly updated during processing. 10, 13

flag (boolean) Flag which permits processing of one ADC sample. 14

iir (struct) Data instance of IIR filter which stores operating environment for the processing function. 9, 13

iir_coeffs[ ] (Q15 fixed-point) Stores IIR filter coefficients in sequential blocks of 6 numbers per stage. Each 2th placeholder of the block (a0 coefficient) is left blank. 11–13, 20

iir_state[ ] (Q15 fixed-point) Stores temporary IIR variables, array is constantly updated during processing. 11, 13

inn (Q15 fixed-point) Temporary variable for incoming sample during RTC mode. 13

is_rt (boolean) Flag which controls entering RTC mode. 13

num_taps (unsigned int 32) Length of FIR filter kernel. 10, 12, 13

num_taps2 (unsigned int 32) Number of active cascaded IIR filter stages. 11–13

outt (Q15 fixed-point) Temporary variable for outgoing sample during RTC mode. 14

pshift (int 8) Scaling factor for IIR filter coefficients. 11

s_period (unsigned int 8) Sampling period (microseconds) for RTC. 13

sampler (Ticker object) The timer interface to attach a function as an interrupt service routine handler. 13, 14

Back end program version history

Version 0.1
Prototype application capable of processing numerical data with fixed FIR filter.

Version 0.2
Prototype application with flexible DSP parameters and numerical processing capabilities.

Version 1.0
Operational firmware with configurable numerical and real-time data processing options.
Fixed-point DSP functions are used instead of floating-point ones.

Version 1.1 (current)
User can specify data block size in numerical mode.
ADC and DAC are made to work directly with fixed-point.

4 Front end application

4.1 Program structure

Front end application is a MATLAB based user interface. Properties of the DSP algorithm are tested by processing a customizable signal, which is stored as an array of samples. By interacting with graphical interface (figures 7 and 8), user can input parameters for the signal as well as specify algorithm coefficients. Connection to the processing device is established by selecting appropriate COM port from a drop-down list. When back-end is connected, user can either send generated signal data to the device or execute RTC mode using specified algorithm. Generated and processed signals are displayed on separate graphs.

Every interactive GUI element is connected with own **callback** procedure, which is executed upon user action. Most of callbacks are used to input numerical data into the program, while few are executing complex functions that are shared between elements, as shown on the figure 9.

Default HANDLES data structure is used to exchange global variables between callbacks. This object is passed as an argument into callback and updated on exit. Because of that, in MATLAB code global variables are seen in form `handles.variable` as members of the object. To shorten this presentation, in this report global variable names simply have preceding dot as **.VARIABLE**, and local variables are displayed as is.

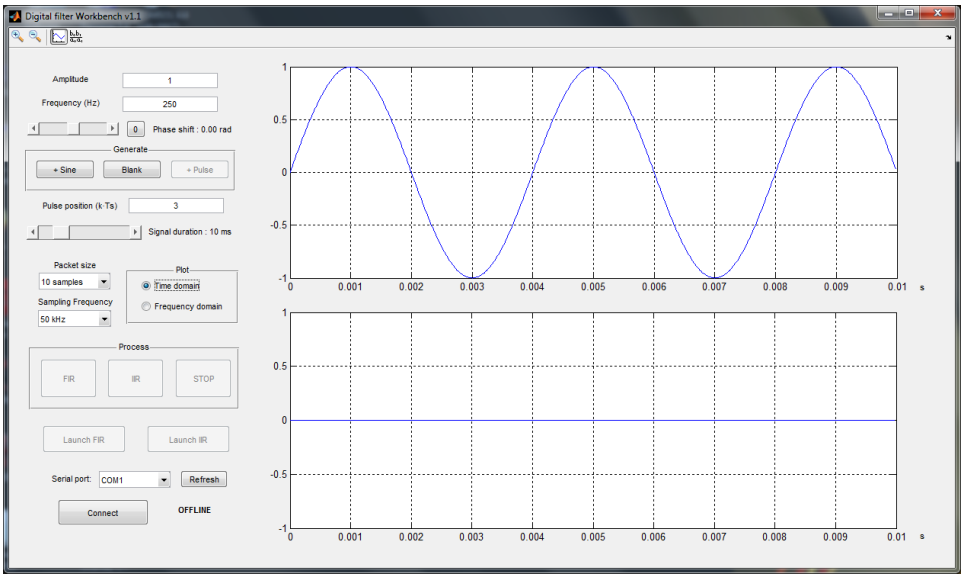


Figure 7: Main application window, signal generator (top left) and graph analyzer (right)

Figure 8: Coefficient editor mode

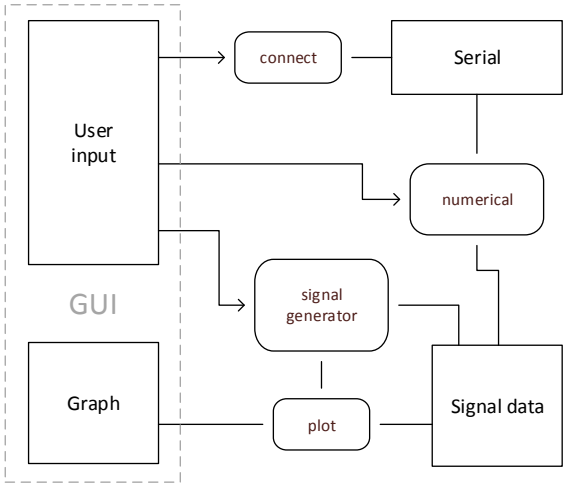


Figure 9: MATLAB application structure

4.2 Signal production and display

Filtering performance can be efficiently demonstrated with a signal that contains rich frequency content. Theoretically, almost any waveform can be approximated with proper combination of sine and cosine waves, or only sine waves with various phase shift. However, infinite bandwidth signals are also required, like digital impulse which is used for analysis of the frequency response. In the program, user is given possibility to construct the signal in two ways:

Sine wave addition: Amplitude, frequency and phase shift are set individually for each new signal. Based on sampling frequency, sequence of time points is created. Parametrized SIN function is evaluated for each time slice, yielding array of discrete values. Output is shown as a line which connects every point in series.

Digital impulse addition: Signal is constructed from several samples of desired magnitude which are added on all-zero line in specified positions. Non-zero samples are shown as vertical spikes of corresponding magnitude and position.

These two methods are essentially different, therefore addition of digital impulses to sine waves is restricted, to prevent irrelevant discontinuities of the signal.

Often, combination of waves should be examined for various sampling frequencies in order to demonstrate aliasing effects. For this purpose, signal

parameters are stored separately from generated data, which allows for perfect reconstruction of the signal for desired sampling rate. Block diagram of complete data production process is shown in figure 10.

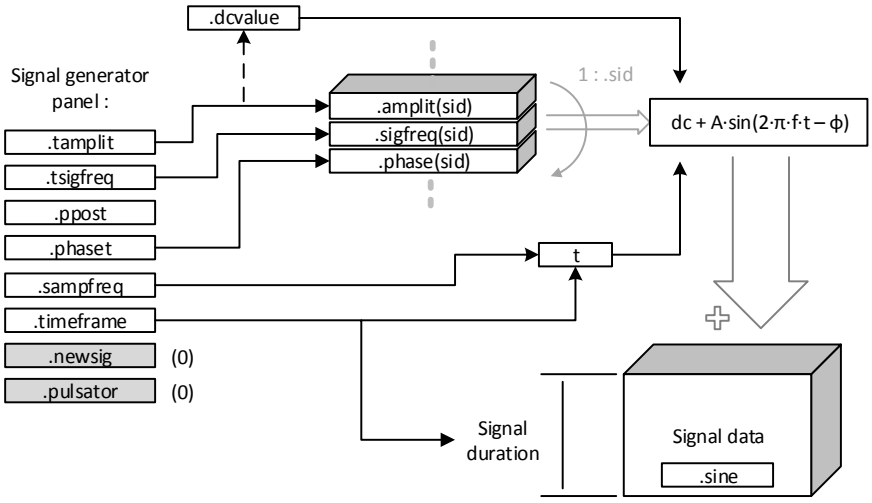


Figure 10: Sine wave addition process

Input fields on the signal generator panel are forwarded to corresponding temporary variable. If last user input is invalid (e.g. frequency is negative), text field callback will restore previous value, giving an error message to the user.

When ADDSINEBUTTON callback is executed, temporary values are saved in arrays by signal index .SID. If wave frequency .TSIGFREQ has been specified as exactly zero, its amplitude .TAMPLIT is added to the total signal offset .DCVALUE. Phase shift is controlled with a slider, and its temporary value is stored in .PHASET.

Array of time points is calculated from duration of the signal .TIMEFRAME and sampling frequency .SAMPFREQ. Every time any of these two variables changes or a new waveform added, DISPLAYER function sets a loop for each .SID element. Sine is created for each set of values from .AMPLIT, .SIGFREQ and .PHASE arrays. All sines are added together with constant .DCVALUE for a vector of time points (equation 5). Generated data is displayed to the user on a top graph AXES1.

$$S = \sum_{n=1}^{n_{max}} \sum_{s=1}^{.sid} \big(A[s]sin(2\pi f[s]t[n] + \phi[s]) + C/.sid\big)\delta[n] \tag{5}$$

Digital impulse function PULSERENDER works in similar way, but uses only amplitude and position fields, .TAMPLIT and .PPOST (figure 11).

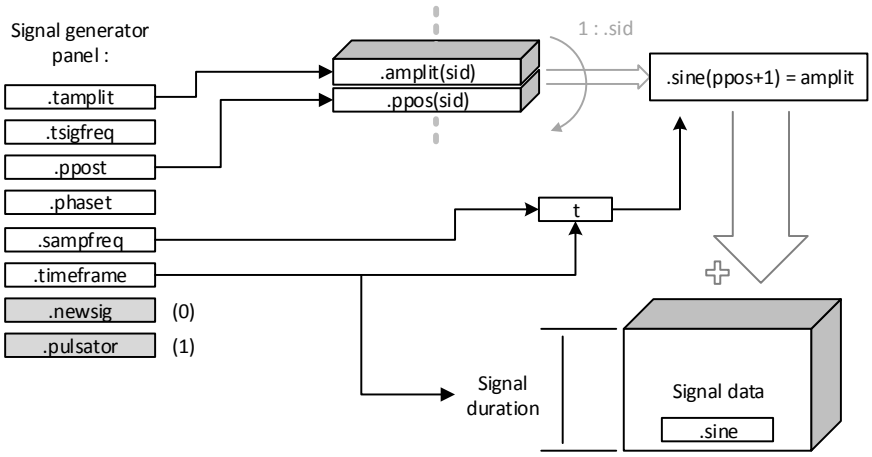


Figure 11: Digital impulse addition process

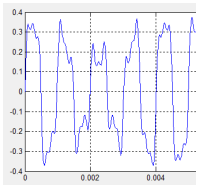
Impulses are set in very similar way, in a .SID loop on a time vector. Position .PPOS is specified in integer of sampling periods, and because of that, .AMPLIT values are directly inserted into blank data array on specified index (equation 6).

$$S = \sum_{n=1}^{n_{max}} \sum_{s=1}^{.sid} A[s]\delta[n] \tag{6}$$

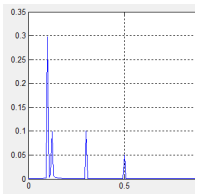
Axillary flags .PULSATOR and .NEWSIG (figures 10 and 11) are global indicators for signal resizing callback FRAMESET so it can pick the proper generator function to reconstruct signal data of different length. Flag .PULSATOR also serves as a switch for drawing function, when it selects between PLOT or STEM display methods.

Signal data can be drawn on a graph directly or in frequency domain, by processing it with FFT algorithm. Function PLOTTER draws signal on the axis by plotting amplitude values against time points. If .DOMAIN is set to 2, signal is fed to FFT and its amplitude scaled accordingly. Full spectrum is plotted against linear frequency from zero to F_s.

Signal can be completely wiped together with all data with BLANKING button callback. .SID index is then set to zero, so signal parameter arrays will be overwritten next time. Data is set to all-zero and top graph is updated. Flag



(a) Time domain signal



(b) Frequency domain signal

Figure 12: Function PLOTTER output

.NEWSIG is set to one so that when duration .TIMEFRAME is changed, data would be automatically filled with zeros.

4.3 Remote procedure call

On application startup, list of available COM ports is obtained from MATLAB serial interface.

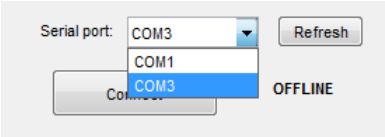


Figure 13: Connection menu

Upon successful connection, SERIAL RPC object is created and pointers to its methods are assigned, providing access to remote functions, described in section 3.3. Flag .ONLINE is set to 1 to aid menu drawing when switching between coefficient and signal generator panels (figures 7 and 8).

Coefficients for FIR and IIR filters are stored in .FIRCOEFFS and .IIRCOEFFS accordingly, as shown on following tables. Variables .FSTAGER and .ISTAGER are necessary to keep track of desired filter order to be implemented.

index	1	2	3	4	5	6	7	8	9	10
coefficient	b0	b1	b2	b3	b4	b5	b6	b7	b8	b9

Table 1: Contents of .FIRCOEFFS array

When user executes any of processing modes on the device, functions FIRKOEFF and IIRKOEFF are translating these vectors directly into FIR_COEFFS[ ] and IIR_COEFFS[ ] arrays on the MBED. Before uploading coefficients onto device, FIR kernel is additionally rearranged in order to match the library function requirements [3].

index	1	2	3	4	5	6	7	8	9	10	11	12
coefficient	b0	-	b1	b2	a1	a2	b0	-	b1	b2	a1	a2
	1st stage						2nd stage					

Table 2: Contents of .IIRCOEFFS array

RTC mode is initiated with either FIRLAUNCH or IIRLAUNCH function. Sampling period .STIMING is transmitted to back end, after which program assumes that MBED has entered real time mode. Communication is then terminated and program is ready to connect to the device again.

4.4 Numerical data processing

Digital samples are sent to back end in packets and processed one packet at a time. Kernel coefficients and state variables (being previous samples) are stored in the firmware, so NUMERICAL function takes care only of data transport. Communication procedure happens in two nested loops, such that inner loops execute transmission within a block of data and outer loop iterates trough blocks of size .PACKET. To keep track of whole range of signal points, local pointer PTR is defined.

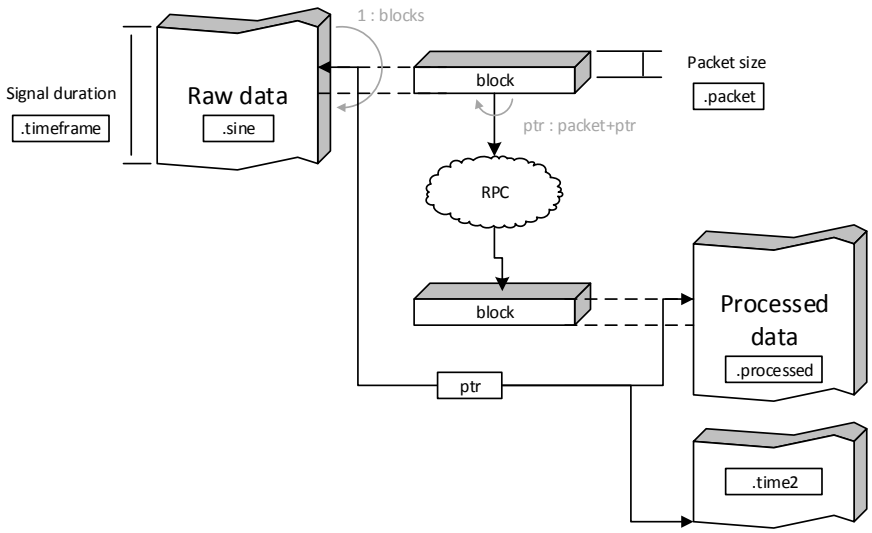


Figure 14: Block diagram of function NUMERICAL

Returned data is stored in .PROCESSED array. Each block that finished processing and came back is immediately plotted on the lower graph. Vector

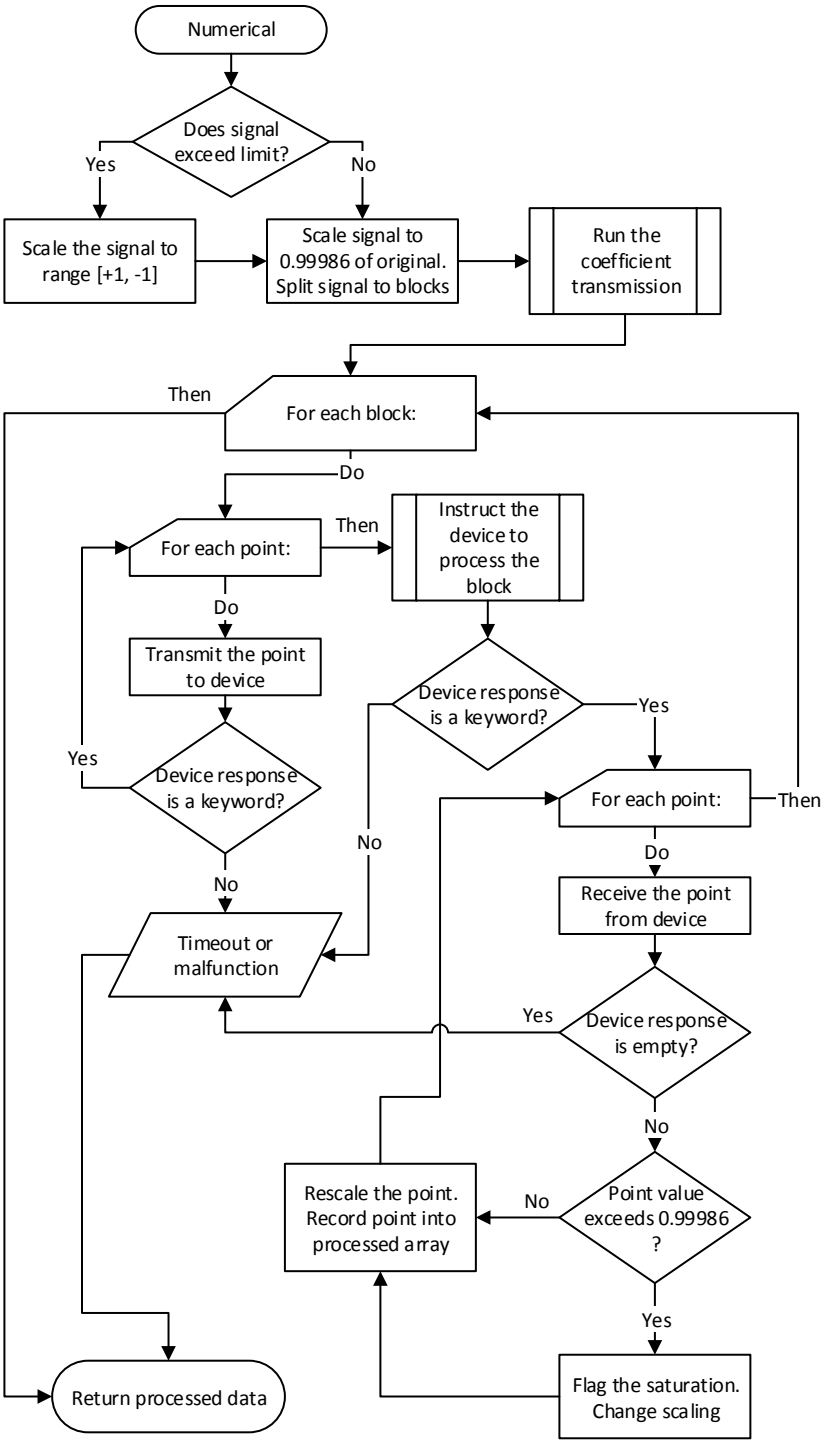


Figure 15: Logic flowchart of data handling

.TIME2 is simultaneously filled with time points to keep the number of non-empty elements equal in both arrays.

Flowchart of data operations is presented on figure 15. To match the dynamic range of desired signal to fixed-point limits, total amplitude is scaled to range from positive to negative one. However, saturation of output accumulator can not be avoided when kernel of the filter has big gain. To inform user about situation, output is additionally tested for saturation, by simply scaling the signal for about one bit lower than maximum. When current filter algorithm output exceeds maximum amplitude of the input, it is definitely saturated. Every back end response is additionally tested for being non empty. Firmware is programmed in such way so that every RPC function return value is non-zero, and otherwise front end assumes serial time out and destroys the connection.

Numerical function checks if user wishes to stop the processing by pressing the STOP button. Callback is then halted, but data is not destroyed and available for further analysis. If user has previously entered just one digital impulse, then .COMPARE flag equals one. In such case, when frequency domain is rendered, FFT result of the initial signal is written into .SPECTRUM to be compared with output spectrum. Resulting frequency response is plotted in decibels, simplifying shape recognition.

testbench.m global variable reference

.amplit Array with amplitudes of generated signals in order of user input. 18

.compare Flag to indicate that processed signal is one impulse, and spectrum comparison of the signals will yield true result. 23

.dcvalue Offset term (direct current) for the signal generator. 18

.domain Indicates the mode of desired data presentation: either time domain (1) or frequency domain (2). 19

.fircoeffs Array with FIR filter coefficients. 20

.fstager Integer with the desired number of active FIR filter stages. 20

.iircoeffs Array with IIR filter coefficients in sequential blocks of 6 numbers per stage. Each 2th placeholder of the block (a0 coefficient) is left blank. 20

.istager Integer with the desired number of active IIR filter stages. 20

.newsig Flag to indicate that graph has been blanked and the generator pool is empty. 19, 20

.online Flag to indicate that the last communication with back end has been successful and device is available. 20

.packet Integer that stores amount of samples per processing block. 21

.phase Array with phases of generated signals in order of user input. 18

.phaset Keeps the temporary phase shift value that is shown in signal generator input. 18

.ppos Array with positions of the impulses in order of user input. 18

.ppost Keeps the temporary pulse position that is shown in signal generator input. 18

.processed Array with processed signal values received from back end. 21

.pulsator Flag to indicate that impulses are being generated. 19

.sampfreq Current sampling frequency in Hertz. 18

.sid Counter of signals that are displayed on the first graph. 17, 18, 20

.sigfreq Array for frequencies of generated signals in order of user input. 18

.spectrum Array with FFT values of the input signal. Used for calculating the frequency response in dB. 23

.stiming Timing period in milliseconds for the RTC operation. 20

.tamplit Keeps the temporary amplitude that is shown in signal generator input. 18

.time2 Array which is used to populate time axis on the second graph. 21

.timeframe Stores the duration of generated signal in milliseconds. 18, 20

.tsigfreq Keeps the temporary frequency that is shown in signal generator input. 17

Front end program version history

Version 0.1
Basic GUI with signal generator and connecting option.

Version 0.2
Added FFT functionality.
Digital impulse option is rewritten, user can model filter kernels.
Coefficient editor mode added as alternate panel on GUI.

Version 1.0
Fully functional release.

Version 1.1 (current)
Packet size can be specified by user.
Added logarithmic impulse response drawing mode.

5 Hardware module

5.1 Circuit logic

To meet the lab requirements of our teacher, we had to consider developing a Printed Circuit Board (PCB) that would meet his expectations and needs for his students. The board has an MBED microcontroller solution (LPC1768) whose CPU clock frequency is 96 MHz. It also contains a voltage divider at the signal input and a reconstruction filter at the output. A rail-to-rail amplifier Application-specific Integrated Circuit (ASIC), TS922 by STMicroelectronics is used as voltage buffer but this unit is however discontinued these days. As a drop-in replacement, MCP6273 by Microchip can be used.

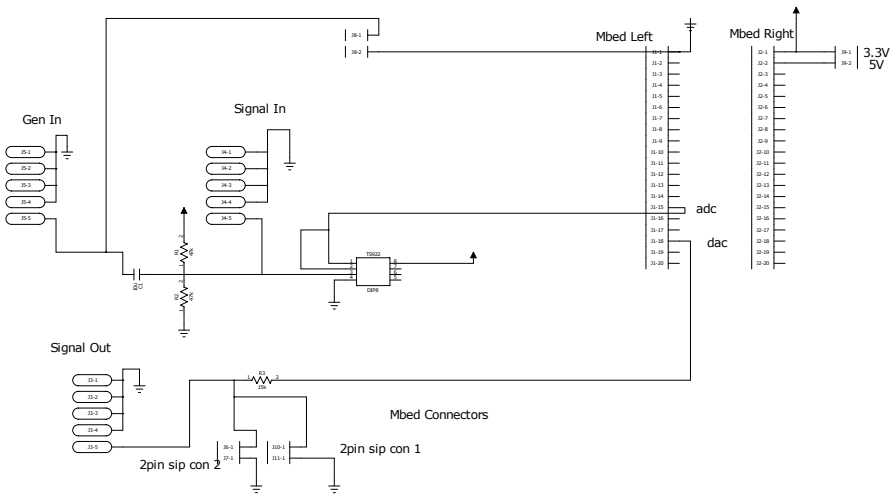


Figure 16: Logic diagram of hardware module

Because the MBED can only accept positive voltage input, it is necessary to add a small coupling and biasing circuit to offset the signal to a midpoint of MBED supply (approximately 1.65 V).

5.2 Board layout

The board contains three I/O BNC-type connectors for easy connection with oscilloscopes, one custom signal input as 2-pin connector, a voltage divider and high pass filter at the input, two twenty pin connectors for the MBED board and a first order passive reconstruction filter at the output. The two 2-sip-pin connectors at the output are meant for the capacitor which students can change depending on what kind of an output filter they want for their sampling frequency. Another 2-pin connector is present, if a 3.3 V or 5 V source is needed from MBED board.

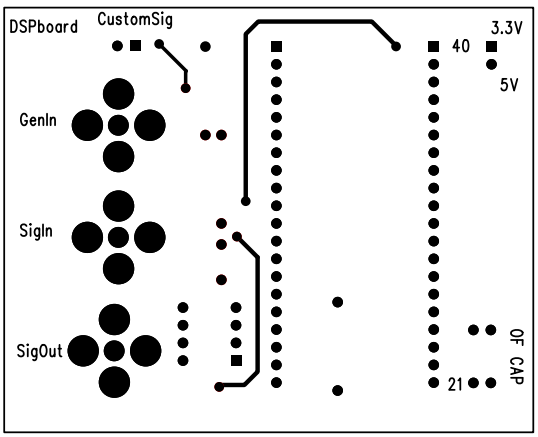


Figure 17: Top layer of the PCB

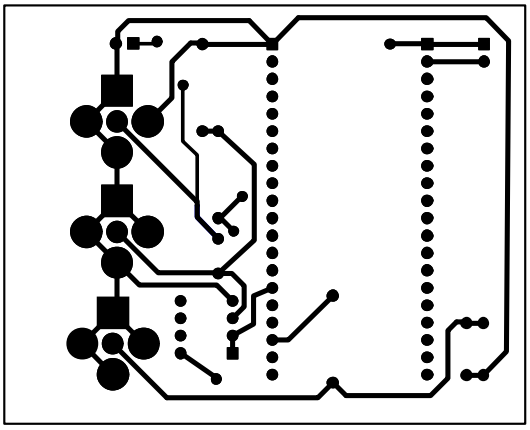


Figure 18: Bottom layer of the PCB

PADS LOGIC was used for the circuit design and PADS LAYOUT for the layout. The CAM files were then exported from the layout file to CIRCUIT CAM envi-

ronment. There the files are processed such that they are compatible with the LPKF milling machine software. To operate the PCB milling machine, LPKF BOARD MASTER is used.

Bill of materials

Reference	Part Name	Description	Tolerance	Value
C1	CAP-CC05,10u,10%	Capacitor	10%	10uF
J6-7 J10-11	CON-SIP-1P	1 Pin connector		
J1	CON-SIP-20P	Mbed Left Con 20 sip pin		
J2	CON-SIP-20P	Mbed Right Con 20 sip pin		
J8-9	CON-SIP-2P	2 Pin connector		
J3	CONRA-5P-200	Signal Output BNC		
J4	CONRA-5P-200	Signal Input BNC		
J5	CONRA-5P-200	Generator Input BNC		
TS922	DIP8	Rail-to-rail, high output current dual operational amplifier		
R3	RES-1/4W,15k,1%	Resistor	1%	15k
R1-2	RES-1/4W,47k,1%	Resistor	1%	47k

Table 3: Bill of materials

6 Results

6.1 EMC scan test

DSP module was tested for electromagnetic emissions in Metropolia UAS laboratory. Following equipment has been used for test:

- Detectus Electromagnetic Compatibility (EMC) scanner with two different antennas
- A Computer with the EMC scanner software
- Rohde & Shwartz GHz oscilloscope measuring the scanner output

Test site is presented on a figure 19. DSP board is placed on EMC scanner table. Its dimensions and position are set in the software.

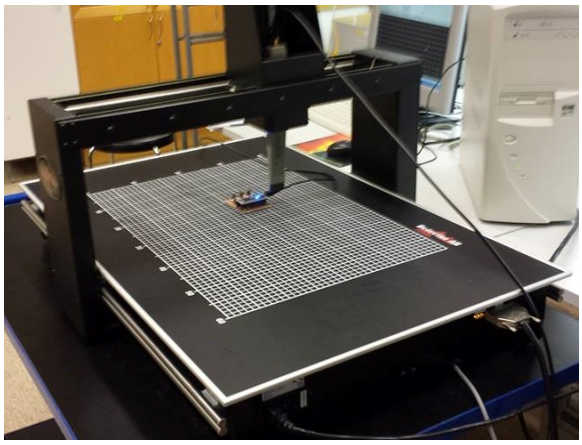


Figure 19: EMC test setup

First scan is made using the 9 KHz - 30 MHz antenna. Results are presented in figure 20. The peak at the beginning is an oscilloscope signal which acts more like a calibrating function. Otherwise we see that there is nothing else notable in the spectrum.

In 30 MHz to 1 GHz measurement again we notice that there is not much happening (figure 21). There is a peak at 941.8 MHz probably due to GSM band waves (high frequency) that are being reflected from the board.

As a conclusion we can safely say that the DSP board does not have any EMC problems. It passes the test even though it does not have a ground plane. The EMC scan was not really required and in practice would not be done to reduce costs.

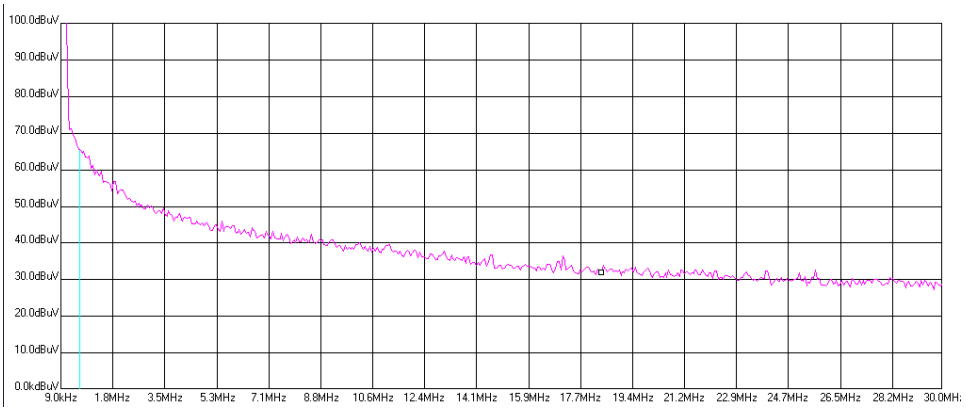


Figure 20: EMC scan results in 9 KHz - 30 MHz band (dBuW against MHz)

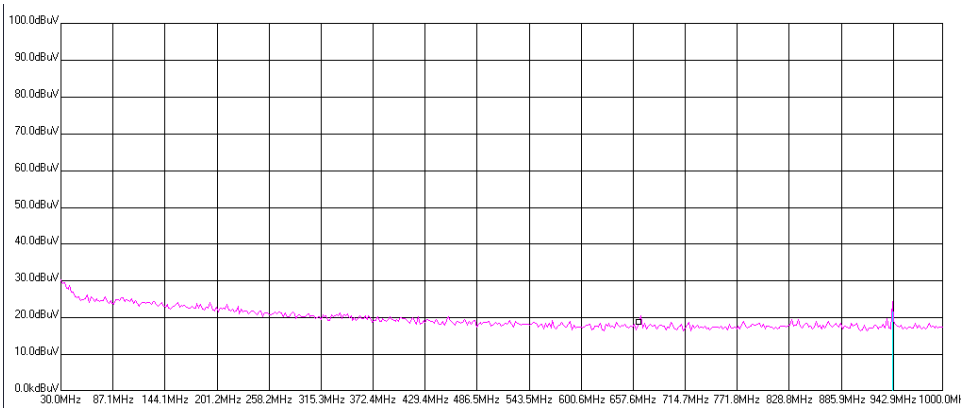


Figure 21: EMC scan results in 30 MHz - 1 GHz band (dBuW against MHz)

6.2 Laboratory works

In total, 13 boards were produced such that the DSP labs could happen in groups of three students.

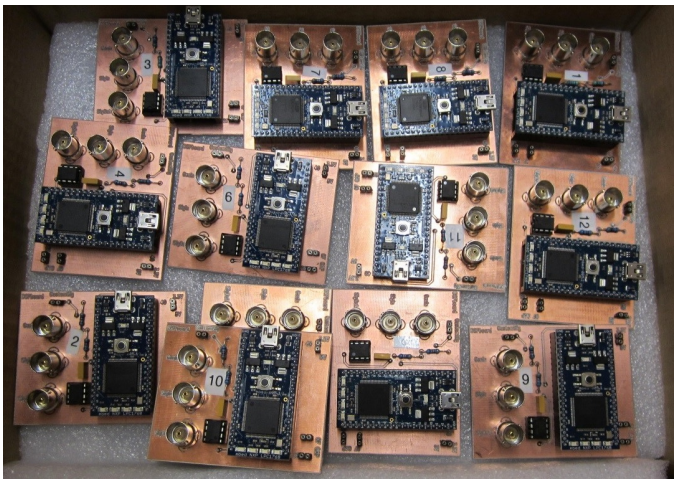


Figure 22: Assembled DSP boards

All in all, the students were excited to use these devices. Once they had gone through the manual they were eager to experiment with different options and see the results. Most agreed that this was a great way of understanding the DSP theory. During the course they were given three labs to complete from the teacher who also required the student groups to submit a report for each lab.

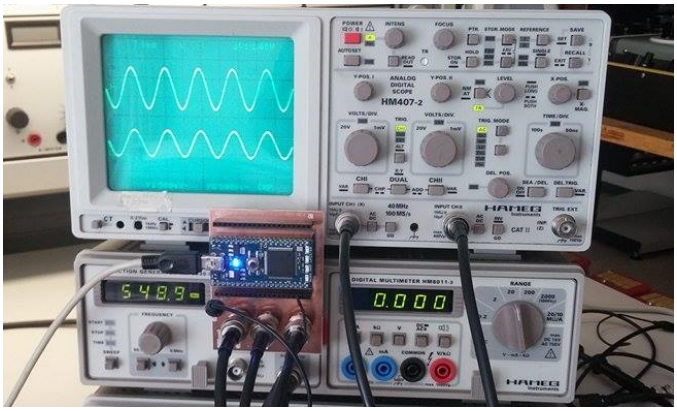


Figure 23: Processor board in RTC mode

7 Conclusion

Simple customizable DSP system with basic computing functionality has been successfully developed, tested and integrated into teaching process. It has been proven that development of low end budget educational platforms for limited use is quite possible.

Next versions of the systems are expected to implement faster ADC and DAC, both of high resolution, most probably situated either on separate ASIC or in specialized DSP device. Front end can be implemented as a standalone program with custom math functionality, preferably using Visual C or Java. Advantage of high-speed binary connection either through USB or LAN is necessary. It is quite possible for system to ascend over DSP libraries by allowing to prototype a kernel from any block diagram constructed by the user.

In the development of DSP systems, one should always consider hardware constrain as well as software development kit flexibility. At high frequencies and data rates, appropriate hardware design details should be considered. As the whole system is teaching oriented, software design has to consider pedagogical issues that come with teaching.

References

[1] Steven W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, California Technical Publishing, 1997.
`http://www.dspguide.com/`

[2] Rob Toulson and Tim Wilmshurst, *Fast and Effective Embedded Systems Design: Applying the ARM mbed*, Newnes, 2012.

[3] ARM, *CMSIS DSP Software Library user manual*, 2014.
`http://www.keil.com/cmsis/dsp`

[4] ARM, *Interfacing with Matlab*, mbed Cookbook, 2014.
`http://developer.mbed.org/cookbook/Interfacing-with-Matlab`