FPGA-Based Implementation for QRS Detection in Electrocardiogram

Ivo Iliev, Lubomir Bogdanov and Serafim Tabakov
Department of Electronics, Faculty of Electronic Engineering and Technologies
Technical University of Sofia
8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria
{izi, lbogdanov}@tu-sofia.bg
{savi}@ecad.tu-sofia.bg

Abstract – **This paper presents an implementation of a QRS detection algorithm in C with custom IP library blocks written in Verilog for a FPGA chip. The hardware design adds support for the processing of the data array that contains the electrocardiographic samples. Focus is put on the software and hardware structure, as well as some performance metrics – the used memory and the time for execution of the processing procedures. Final results shown that the hardware implementation is slower than the microprocessor-only implementation, its ROM usage is 700 bytes less. The RAM usage is equal in both cases. The hardware IP would also allow for other devices in the system to get more microprocessor time (e.g. interrupt handling)**.

Keywords – **FPGA-based ECG; digital filter; electrocardiogram; embedded systems; FPGA; Verilog.**

I. Introduction

The electrocardiogram (ECG) is one of the most commonly used diagnostic procedures in the medicine. It is a fact that modern electronic devices for recording of heart activity are subject to intensive development in order to increase their diagnostic capabilities. Typical examples are connected with the requirements to minimize size and power consumption in specific applications, such as in long-term monitoring, remote monitoring, wearable applications, etc. In recent years, the attention of researchers and developers has increasingly turned to Field Programmable Gate Array (FPGA-based) hardware implementations. The main reason is the presence of a serious computing resource in these architectures, allowing parallel processing of data in real time. In particular, for the processing of ECG signals, such applications prove their applicability, especially in the implementation of interference filtering algorithms (base-line drift, power line interference, electromyographic noises, etc.), subsequent detection of QRS complexes, recognition of rhythm abnormalities, shape analysis, etc. In one of the first publications [1] devoted to the application of FPGA for ECG signal processing, the authors present algorithm performing QRS complexes detection and beats classification. The advantage of parallelization in the processing of the data by the hardware and software part of the system is evaluated, allowing a significant increase in speed, even at a lower clock frequency. In later publications, the authors present FPGA-based solutions for both rhythm analysis [2], [3] and digital ECG signal processing applying least-square linear phase finite impulse response (FIR) filter [4], least-square FIR filter [5], Hermite polynomials for ECG signal characterization [6]. It should be noted that the majority of FPGA applications are aimed at performing a separate stage of the overall process of ECG signal registration.

In the focus of this work is an approach different from existing implementations because it uses both a microprocessor and a programmable logic. This is appropriate for devices that implement more than one function – such as ECG processing, visualizing the data on a graphic display and sending the data through a communication interface. In addition, some metrics related to the required memory and signal processing time were evaluated.

II. Method

The block diagram of the hardware is shown in Fig. 1. The FPGA chip has many integrated modules, however, only the blocks that are used in described below procedures are shown in Fig. 1. A demo board from Digilent is being used. The model is Zybo (first revision) and it is populated with a Xilinx FPGA part number XC7Z010, from the Zynq family. This FPGA has a 12-bit A/D converter which makes it suitable for a final implementation of an ECG device. The FPGA comes integrated with two ARM Cortex A9 microprocessors, as well as many hard-wired modules such as USB, Ethernet, SD card module, GPIO, UART, I2C, etc.

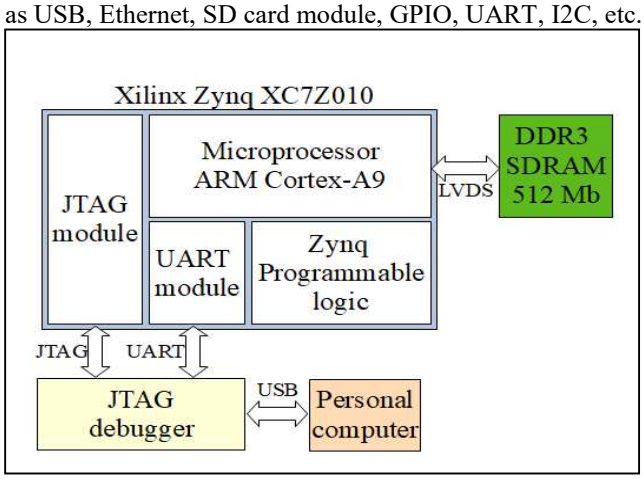

Fig. 1. Block diagram of the proposed hardware for QRS detection. Auxiliary modules not shown for clarity.

The developments in this study are supported by the Bulgarian National Science Fund, grant КП-06-Н37/9.

This part of the FPGA is called by the manufacturer as a "processing system" (PS) and is basically a full-featured microcontroller without memories (RAM and ROM). Next to it, a "programmable logic" (PL) part of the chip could be found. This is the classic FPGA containing complex logic blocks (CLB), digital signal processor blocks (DSP) and block random access memory (BRAM) cells. To reduce on-chip resource spending from the PL part, an external DDR3 RAM is used. It contains the C program to be executed, as well as the input array of samples. The purpose of making an ECG block on an FPGA is to allow for rapid development of new designs – engineers who are not familiar with the specific data processing can just drag and drop a custom ECG module. DDR SDRAM is connected through a hard-wired memory controller that allows up to 1 GB external memory to be connected with speeds of up to 1333 Mb/s. If the design is to be implemented as a working device in the future, an option to connect external non-volatile memory exists – the processing part of FPGA contains a NAND/NOR/QSPI flash interfaces for adding external ROM memory. The current hardware solution uses the DRAM for instruction and data storage during a single debug session that is enough for implementing the algorithm and performing the tests.

III. Algorithm of Test Procedure for QRS Detection

An algorithm consisting of two parts (digital filter for ECG signal de-noising and QRS detector), was developed, implemented and applied for processing a signal from the MIT-BIH database [7]. The consecutive steps of the algorithm are shown in Fig. 2.

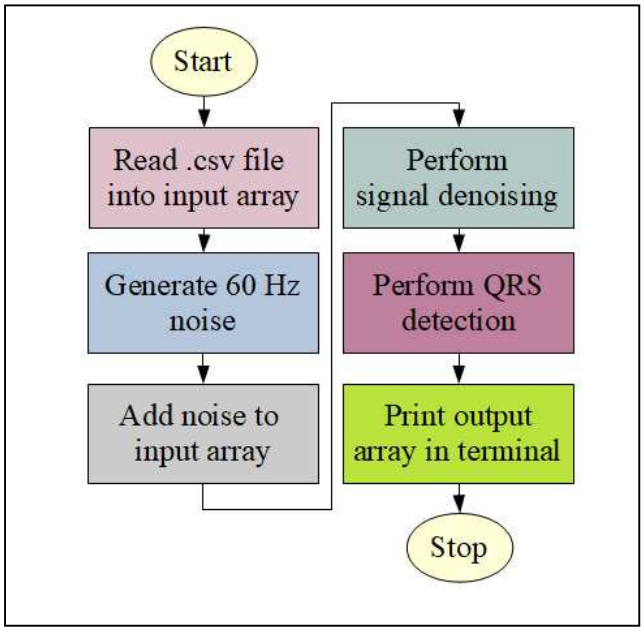

Fig. 2. Block diagram of the test procedure for QRS detection.

The first step is to download the selected ECG signal from the MIT-BIH database. To extract the data, one must install the WFDB software package. It must be noted that this package is open-sourced and stand-alone, and therefore no MATLAB installation is needed. The chosen OS in this case was Ubuntu Linux 20.04. The required dependencies for Linux can be installed from a terminal with the command:

```
apt install gcc libcurl4-openssl-dev libexpat1-dev
```

For the current work, the file number 200 was chosen. To convert the raw data "200.dat" file to a comma-separated value (CSV) text file, the following command has to be invoked:

```
rdsamp -c -r mitdb/200 > 200.csv
```

It is important to note the parameters of the measurement setup that sampled these signals because the processing procedures are developed according to them. The input range of the A/D converter is a bipolar +/- 5 mV voltage, the resolution of the A/D converter is 11 bits, therefore the integer outputs will be in the range 0 – 2047, where the zero is represented with 1024. Each bit weight is 4.883 uV, and the signal is sampled 360 times per second (360 Hz). The mains frequency is 60 Hz.

The second step of test procedure includes writing a C program that performs the actual ECG processing. The Eclipse integrated development environment (IDE) is used for this purpose. The toolchain is GCC. To evaluate the effectiveness of the filter procedure to remove power line interference, a sinusoidal signal with a frequency f_{mains} = 60 Hz and an amplitude as large as the reference signal is superimposed on the ECG signal from the database.

$$pl(T) = \sin\left(2\pi \frac{f_{mains}}{f_{ecg}} T\right) . N_{ADC}, \quad (1)$$

where: f_{ecg} is the sample rate of the A/D conversion, N_{ADC} is the resolution, T is the sampling period. Afterwards, the points from the pl[ ] array are added to the ecg[ ] input array:

$$ecg\_pl[T] = ecg[T] + pl[T] \quad (2)$$

The filtering procedure for ECG signal de-noising is thoroughly described in [8]. The used "FilterDxN" was designed implementing the simple principle of moving averaging of N signal samples distanced each other by D samples. The filter has a comb frequency response characteristic with a high-pass cut-off defined by N and zeros at the integer ratio of the sampling frequency fs divided by D. According to the signals included in the MIT-BIH database we chose the values for D = 6 and N = 42. Substituting these values into formulas 3 and 4 gives a first zero at a power line frequency of 60 Hz and a cut-off frequency f_{3db} of approximately 1 Hz.

$$D = \frac{fecg}{fmains} \quad (3)$$

$$N = \frac{3}{4}\left(\frac{fmains}{f3db}\right) \quad (4)$$

After the successful filtering of the ECG signal, the QRS complex detection algorithm described in [9], with adaptive amplitude and slope criteria for R-peak recognition, is applied.

To increase the processing speed of the design, the algorithm uses only integer values. This makes the program easier to implement on microarchitectures without a floating point unit (FPU). As a side test, the same algorithm was written with floating point variables and the processing was run again. There were no differences in the output result between the integer and floating point versions.

IV. Porting the Algorithm in C to the FPGA

The third, and final, step was to port the existing C code to an FPGA. To do this, the first test is to make the original C code work on the FPGA without custom intellectual property (IP) blocks. The synthesis of the 60 Hz mains signal and its addition to the reference signal can be skipped. The resulting reference-plus-mains array is generated from the original C program with the help of the printf( ) function:

```
printf("uint8_t ecg_pl[] = {\n");
for(int i = 0; i < RECORD_LENGTH; i++){
printf("%d, ", ecg_plus_power_line[i]);
        if((i % 8) == 0){
                printf("\n");
        }
}
printf("};\n");
```

The STDOUT output is written to a header file and included in the FPGA project. A GPIO module was added to the programmable logic part of the FPGA. It was used for timing measurements. A UART module was initialized from the processing system part of the FPGA. It was used for printing the output buffer. The design showed exactly the same results as the results from the original C program performed by a PC. Next, the hardware IP modules had to be developed.

To create a custom IP, the Vivado IP integrator can be used [10], [11]. Most of the top-level Verilog descriptions are written automatically by Vivado. The only part that has to be written by the user is the HDL code of the IP block itself [12], [13]. There are not enough CLB to synthesize the whole detector – the current device has 4400 slices, and the DxN filter itself takes 4600 slices. So, only parts of the signal denoising and parts of the QRS detection from the original C program will be implemented with hardware. The final design will be a mix of the microcontroller firmware and small FPGA accelerators.

The first hardware IP that is to be implemented is an averaging circuit from the DxN filter that would replace the following code in C:

```
m = 0;
for(int j = -21; j < 21; j++){
        m = m + ecg_plus_power_line[i + (j * 6)];
}
m = m / 42;
```

where “i" is the current sample that is to be processed by the filter algorithm. The corresponding Verilog code is:

```
always @(posedge clk)

begin:B1
if(control_reg[0] == 31'h01)
begin
    status_reg[0] = 31'h00;

    sum =
input_reg[0]+input_reg[1]+input_reg[2]+
input_reg[3]+input_reg[4]+input_reg[5]+
input_reg[6]+input_reg[7]+input_reg[8]+
input_reg[9]+input_reg[10]+input_reg[11]+
input_reg[12]+input_reg[13]+input_reg[14]+
input_reg[15]+input_reg[16]+input_reg[17]+
input_reg[18]+input_reg[19]+input_reg[20]+
input_reg[21]+input_reg[22]+input_reg[23]+
input_reg[24]+input_reg[25]+input_reg[26]+
input_reg[27]+input_reg[28]+input_reg[29]+
input_reg[30]+input_reg[31]+input_reg[32]+
input_reg[33]+input_reg[34]+input_reg[35]+
input_reg[36]+input_reg[37]+input_reg[38]+
input_reg[39]+input_reg[40]+input_reg[41];

        div_sum = sum / 42;

        output_reg = iterator_reg - div_sum;
    end
end
```

The second hardware IP is an averaging function from the ECG detection algorithm:

```
int avg(int *arr, int size){
        int sum = 0;
        int i;

        for(i = 0; i < size; i++){
                sum += arr[i];
        }

        return sum / size;
}
```

and because this function is invoked twice, there are two instances of the Verilog IP in the design. The Verilog code equivalent is:

```
always @(posedge clk)
begin:C1
    id_reg = 32'h0000000d;

    if(control_reg[0] == 32'h01)
    begin
       status_reg[0] = 32'h00;

        sum <= input_reg[0] + input_reg[1] +
               input_reg[2] + input_reg[3];
        output_reg <= sum >> 2;
        status_reg[0] = 32'h01;
    end
end
```

The resulting FPGA design is shown in Fig. 3. This is a block design view from Vivado. The ZYNQ Processing system, Processor System Reset, Advanced Extensible Interconnect (AXI) crossbar switch and AXI GPIO blocks are standard IP blocks from Xilinx. The DDR memory is off-chip, that is why it is not shown in the diagram. The system frequency is 100 MHz but could be lowered, if needed. This is the frequency at which the hardware IP blocks would be running also.

After the successful synthesis, a driver has to be written in Xilinx Vitis to expose the custom IP input/output registers to the firmware of the ARM Cortex-A9 microprocessor. To

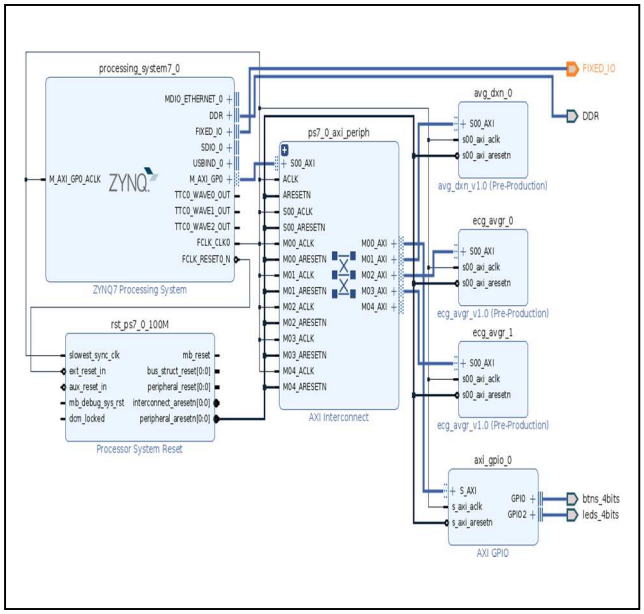

Fig. 3. Block diagram of the proposed hardware.

do this, a structure in C has to be mapped to the corresponding registers from the register map with the following code (only the DxN averaging circuit is given, the code for the ECG blocks is analogous):

```
typedef struct {
    uint32_t control_reg;
    uint32_t status_reg;
    uint32_t iterator_reg;
    uint32_t input_reg[10];
    uint32_t output_reg;
}averager_d_x_n_t;
#define AVERAGER_D_X_N_0 ((averager_d_x_n_t*)
XPAR_AVG_DXN_0_S00_AXI_BASEADDR),where the base address macro for this block XPAR_AVG_DXN_0_S00_AXI_BASEADDR is generated by Vivado, and currently is 0x43c0.0000.
```

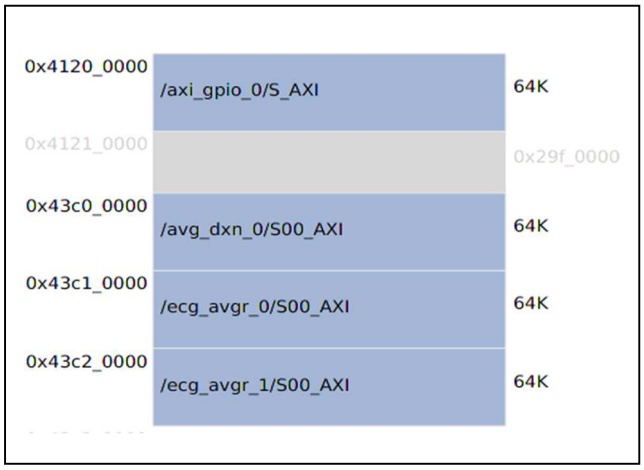

Fig. 4. Memory map of the AXI peripherals in the system.

The memory map of the AXI peripherals in the system is shown in Fig. 4. The two instances of the ECG blocks are at 0x43c1.0000 and 0x43c2.0000. These addresses are fixed and cannot be changed after the synthesis is complete. The addresses of the registers do not change during the signal processing but the values of these registers can be loaded with new samples arbitrarily. Only one signal from the test data set can be processed at a time. The DDR and the internal UART are not part of this map, because Vivado shows only the synthesized blocks there.

V. Results

The final firmware and bit-stream file were downloaded to the target FPGA. An oscilloscope probe was connected to one of the GPIO pins to help with the timing measurements. The final results are shown in Table 1 and in Fig. 5. The "ecg" plot is the reference signal, the "ecg_pl" is the reference signal with added power line noise of 60 Hz, the "ecg_filter" plot is the filtered signal with the DxN filter, the "qrs" plot shows the detected R-peaks.

A comparison with a non-FPGA implementation consisting of a NRF52832 microcontroller, operating at 64 MHz was also made. The chosen microcontroller has a Bluetooth Low Energy (BLE) module that makes it suitable for portable ECG detection. The chip integrates a Cortex-M4 microprocessor with 512 kB of Flash memory and 64 kB of SRAM.

Although the hardware implementation is slower than the no-custom IP implementation, it is faster than the microcontroller-only one. The RAM usage is equal in both FPGA cases because the algorithms share the same data buffer. The hardware IP would allow for other devices in the system to get more microprocessor time (e.g. interrupt handling), so the overall utilization of the system resources may be better with the custom IP blocks. The design could be implemented as a single block on more advanced FPGA and be used for the purpose of medical devices.

Table 1. Design Performance and Resource Consumptions (1000 samples or 3.846 sec window)

	NRF52832 DxN+ QRS	**FPGA, No IP, DxN+ QRS**	**FPGA, With IP, DxN**	**FPGA, With IP, QRS**
ROM, bytes	21240	9596	3876	5864
RAM, bytes	32888	33936	33936	33936
Exec, ms	31.5	1.516	0.412	8.240

VI. Conclusion

The presented work in this paper shows an implementation of a QRS detection algorithm on an FPGA device. The original algorithm was ported from C to Verilog and some timing tests were performed. The output results

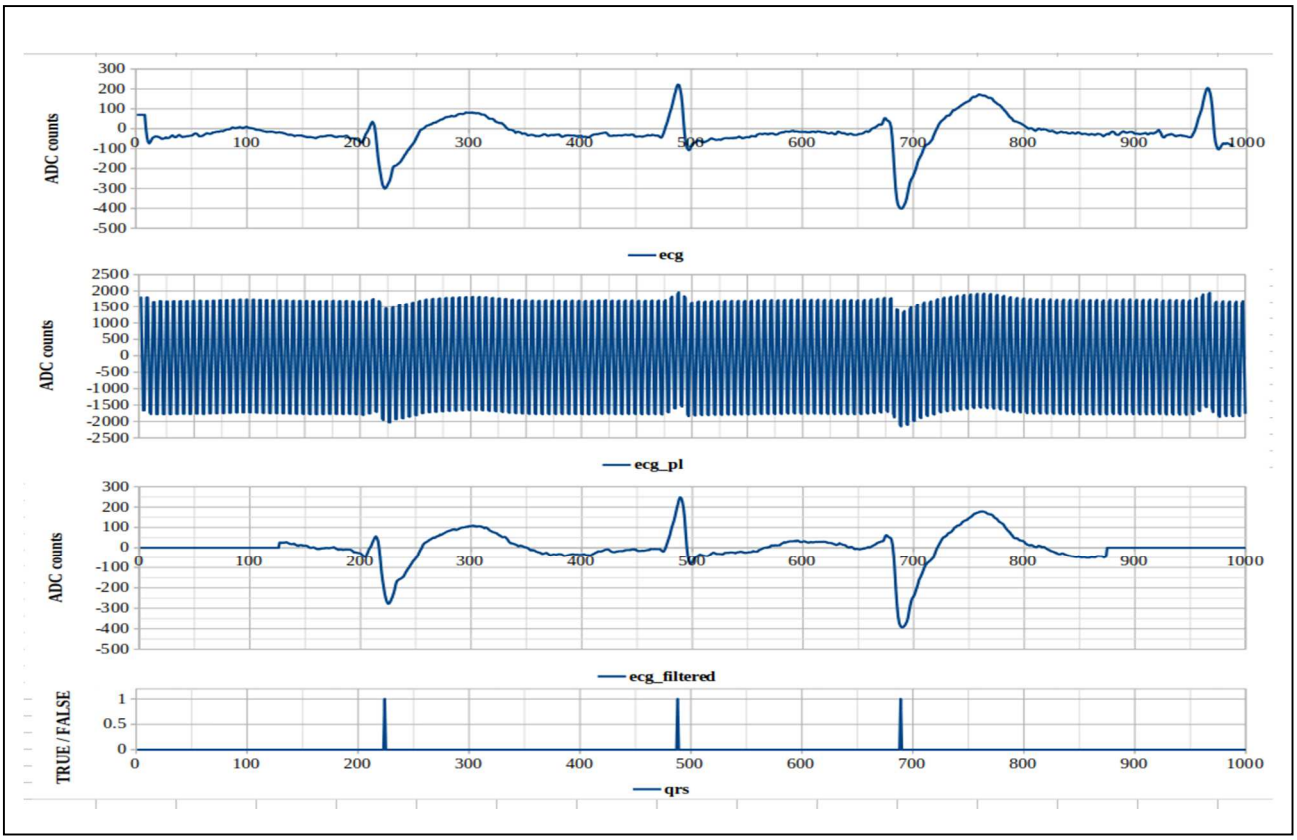

Fig. 5. Results from the FPGA design implementing a QRS detection algorithm.

are accurate and coincide with the ones from the original program. Future improvement of the design might include better Verilog custom IP libraries to increase the speed of the detection. The target FPGA device could be changed with a better one, with more resources. This would allow more parts of the original code to be implemented in the hardware. The design could be tested along other running software/hardware like screen updating, communication with a remote host, keyboard scanning, etc.

ACKNOWLEDGMENT

The authors are thankful for the support provided by the Bulgarian National Science Fund under Grant Ref. No. КП-06-H37/9 “Audio transformation (sonification) of the electrocardiogram – a new approach for remote monitoring of the heart activity”

REFERENCES

[1] M. Cvikl, A. Zemva / Digital Signal Processing 20 (2010) 238–248

[2] L. V. Rajani Kumari, Y. Padma Sai, N. Balaji, K. Viswada, "FPGA Based Arrhythmia Detection", 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015), pp.970-979, Elsevier, 2015.

[3] Karataş, F.; Koyuncu, İ.; Tuna, M.; Alçın, M.; Avcioglu, E.; Akgul, A. Design and implementation of arrhythmic ECG signals for biomedical engineering applications on FPGA. *Eur.* Phys. J. Spec. Top. 2022, 231, 869–884.

[4] M. G. Egila, M. A. El-Moursy, A. E. El-Hennawy, H. A. El-Simary, A. Zaki, "FPGA-based electrocardiography (ECG) signal analysis system using least-square linear phase finite impulse response (FIR) filter", Journal of Electrical Systems and Information Technology 3, pp. 513–526, 2016.

[5] K. Krishnan, K. Prabhu, S. Manimaran, M. Rajan, FPGA Based electrocardiogram (ECG) signal analysis using linear phase finite impulse response filter, International Research Journal of Engineering and Technology (IRJET), Volume: 07, Issue: 03, pp.873-876, e-ISSN: 2395-0056, 2020.

[6] M. P. Desai, G. Caffarena, R. Jevtic, D. G. Marquez, A. Otero, A Low-Latency, Low-Power FPGA Implementation of ECG Signal Characterization Using Hermite Polynomials, Electronics 2021, 10, 2324, https://doi.org/10.3390, 2021.

[7] G. B. Moody, R. G. Mark, "The impact of the MIT-BIH Arrhythmia Database", IEEE Engineering in Medicine and Biology Magazine, pp. 45-50, PMID: 11446209, DOI: 10.1109/51.932724, 2001.

[8] Iliev I, Tabakov S., Krasteva V., Combined High-Pass and Power-Line Interference rejecter Filter for ECG Signal Processing, Proceedings of the Technical University – Sofia, vol. 58, book 2, pp. 7-13, 2008. ISSN 1311-0829

[9] Iliev I., V. Krasteva, S. Tabakov, 2007, Real-time detection of pathological cardiac events in the electrocardiogram, Physiological Measurement, 28, 259-276.

[10] Creating and using custom IP blocks both in Verilog and using High-Level Synthesis, University of South Florida, online, 2019.

[11] Vivado Design Suite User Guide: Creating and Packaging Custom IP, User guide UG1118, Xilinx Inc., online, 2023.

[12] 7 Series DSP48E1 Slice, User guide v1.10, Xilinx Inc., 2018.

[13] Vivado Design Suite Tutorial - Creating and Packaging Custom IP, User guide UG1119 v2021.2, Xilinx Inc., online, 2021