INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 3, PP. 1-8
Manuscript received May 16, 2025; revised July 2025. doi: 10.24425/ijet.2025.153627

FPGA implementation of normalized correlation function

Krzysztof Mroczek

***Abstract*—Correlation analysis is a frequently used tool in signal detection and classification tasks. This paper presents the design and FPGA implementations of a hardware module for calculating the Pearson correlation coefficient. This module is designed for use in signal template matching, where a measurement signal is correlated with a template. It has been described in Verilog and implemented on Intel Cyclone V FPGA. The module consists of two main parts, which are: a correlation filter and normalization modules. Correlation filters performing the calculation in the time domain and in the frequency domain are described. The project has been verified in simulation using ModelSim and checked on hardware. As a result of this work, hardware IP cores are developed enabling parametrization and programming in data word-lengths, filter size, calculation speed, FFT/IFFT size, length, and number of processing templates. Developed resources are intended to be used in FPGA-based hardware, e.g. DAQ systems, working with sampling frequencies from kHz to above 130 MHz.**

***Keywords*—correlation; hardware algorithms; FPGA; embedded systems; time series analysis; pulse recognition**

I. Introduction

DETECTION of signals with specific features in a stream of samples is a common issue in many scientific and industrial applications. Similarity degree between signals can be estimated using cross-correlation function. Correlation methods were proposed for signal recognition and detection, measurements of signal delays, physics experiments [5] - [12]. Cross-correlation measure allows detection of signals in noisy channels. This work considers the normalized version of cross-correlation calculated according to the Pearson formula [1, 3]. The time dependent Pearson coefficient can be assumed as a measure of linear similarity strength between two set of samples. Values of this coefficient are within the range {-1,1}. The range does not depend on the correlated signals' sizes. The closer the coefficient value is to 1, the greater the correlation. A negative -1 value means that the correlation is linear perfect too but the increase of one signal value is correlated to the decrease of the value of other signal and the vice versa [1, 14].

This paper presents the design and FPGA implementation of a hardware module for calculation of the Pearson coefficient. The Pearson coefficient is in this text denoted as NCC. This module calculates NCCs between the measure signal and templates in every sliding window taken from input stream. Two FPGA-based IP cores for NCC calculation are described. First IP core, denoted as *ncc_td*, includes a cross-correlation filter designed in the time domain. This matched filter calculates non-normalized cross-correlation defined by the numerator of the Pearson formula. Next, the filter output data are normalized to NCC values using a normalization module. Architecture with matched filter enables achieving low computation latency. The term *latency* means the number of clock cycles beginning from the clock cycle in which a new sample is entered into the IP core input and ending in the clock cycle in which NCC result for time-window ending on this new sample is available on the IP core output. FPGA resource requirements depends on the size of the correlation filter. To achieve the assumed number of clock cycles for calculation, the number of multiplication modules can be selected accordingly. Therefore, resources usage can be limitation for high-throughput processing.

The second IP core, denoted as *ncc_fd*, implements cross-correlation filter in the frequency domain using overlap-save FFT/IFFT algorithm. The non-normalized correlation data are then normalized in similar way as in the *ncc_td*.

This paper is organized as follows. Section II describes the method applied for implementation of NCC on hardware. Section III presents the design of the main module of the *ncc_td*. Section III.A describes correlation filters developed in this work. Modules developed for NCC normalization are described in Section III.B. Implementation results for Intel Cyclone V FPGA are presented in Section III.C. Section IV describes the design of the *ncc_fd* and its implementation results. Section V reports verification of designed IP cores and discusses achieved results. Finally, section VI concludes this paper.

II. NCC Calculation for FPGA Implementation

The Pearson correlation coefficient (NCC) of two data vectors f and s is defined as [1, 3]:

$$NCC = \frac{\sum_{i=0}^{N-1}(f(i)-\mu_f)(s(i)-\mu_s)}{\sqrt{(\sum_{i=0}^{N-1}(f(i)-\mu_f)^2)(\sum_{i=0}^{N-1}(s(i)-\mu_s)^2)}} \quad (1)$$

where N is the vectors size, μ_f, μ_s are means of f and s.

One of the main project assumptions is to determine how the input vectors can vary for subsequent NCC calculations. If no data sharing between the subsequent vectors is assumed, NCC computation can be performed by applying an acceleration of arithmetic operations in (1). [14] describes NCC accelerator for databases and data center applications. In this project, data from M = 16, …, 64 input vectors (elements of vectors can be 32, 16 or 8-bit) are entered into the computation module in blocks of

Krzysztof Mroczek is with Institute of Radioelectronics and Multimedia Technology (IRTM), Warsaw University of Technology, Poland (e-mail: krzysztof.mroczek@pw.edu.pl).

64 bytes. The computation module consists of the ACC engine for computing the sum of products, sum of elements and sum of squares in (1) and the COEFF engine for calculation of NCC values. The accelerator computes M(M-1)/2 NCC values for every pair of input vectors. The computation time depends on the vectors sizes and DMA throughput.

In design described in this paper, it was assumed that *f* is a vector taken from input stream, *s* is a template vector which is unchanged during processing. These conditions are typical for measuring of similarity to a pattern in time series of samples. The transformation of formula (1), applied in IP cores described in Section III and IV, is as follows. The numerator in (1) is rewritten as:

$$NUM(k) = L(k) - LCM(k) - LC(k)$$

Where:

$$L(k) = \sum_{i=0}^{N-1} f(i+k)\, s(i) \qquad (2)$$

$$LCM(k) = \mu_s \sum_{i=0}^{N-1} f(i+k) \qquad (3)$$

$$LC(k) = \mu_{fk} \sum_{i=0}^{N-1} (s(i) - \mu_s) \qquad (4)$$

where *k* is the position of sliding window in input stream, μ_{fk} is the mean in the k-th position. The denominator in (1) is calculated using running sum and running sum of squares (5):

$$DEN(k) = \sqrt{W(k)PS} \qquad (5)$$

$$W(k) = \sum_{i=0}^{N-1} \left(f(k+i) - \mu_{fk}\right)^2 = \frac{-1}{N}\left(\sum_{i=0}^{N-1} f(k+i)\right)^2 + \sum_{i=0}^{N-1} f^2(k+i) \qquad (6)$$

where *PS* is the pre-calculated sum of squares for template in (1).

L(k) is the equation describing correlation filter, known as matched filter. The computational cost is N multiplications and N - 1 additions per one result. To calculate the running sum and the running sum of squares in (6), four additions are required. Calculation of W(k) requires five additions and three multiplications per one signal window. LC(k) can be assumed equals to 0 if μ_s is calculated without significant rounding errors, caused by division by *N*.

In this work, several configurations of the NCC hardware module have been developed. The following features can be configured:

- choosing of IP-core architecture based on correlation filter,
- template size N: constant or programmed in run-time,
- word-widths of samples and templates,
- number of templates $n_s >= 1$; NCC modules enable calculation of correlation coefficients for selected numbers of templates,
- balance throughput vs. resource usage,
- preloaded or programmed template data.

I developed the described hardware components in Verilog (with some SystemVerilog extensions). Verilog coding for FPGA implementation provides full flexibility in description of a structure of a design. FPGA CAD programs like Quartus Prime, Vivado include tools for automatic HDL generation of IP instances. Verilog constructs `define, parameter, localparam, generate enable code parametrization. Parametrization enables setting of design parameters in compile time, including: word-widths, latencies, number of processing elements, components instantiations, and others.

III. THE NCC_TD DESIGN

Architecture of the main module of the *ncc_td* core, denoted as *ncctop*, is depicted in Fig. 1. The *ncctop* performs calculations according to formulas (2), (3), (5), (6) in pipeline architecture. FIR correlation filter is used to calculate L(k). Samples read from an input buffer are entered into the filter input (*smpl_in*). A new sample can be entered as valid (*smpl_valid* = 1) when ready signal (*smpl_rdy)* is set active. The time period between adjacent valid strobes must not be less than the filter calculation time.

Two architectures of correlation filter have been developed, described below:

NCC_FIP

The instance of Intel FIR II IP core is used for L(k) calculations. Template data can be loaded in run-time or pre-loaded. Filter parameters are configured in compile time, what includes: filter size (*N*), number of processed templates, number of clock cycles per calculation of L(k) (S_t).

NCC_MEM

This is the programmable filter described in Section III.A. Templates data, templates size (*N*), number of clock cycles per calculation of L(k) (S_t), and number of processed templates (n_s) are programmed in run-time.

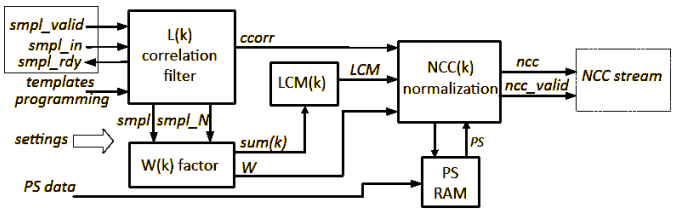

Fig. 1. Block diagram of the *ncctop* module

Correlation filters support integer and fixed-point formats for samples and coefficients. Every new sample from *smpl_in* input is also entered into the input of the W(k) module. If NCC_MEM architecture has been selected, the sample delayed by *N* sample periods (*smpl_N*) can be used for calculation of running sums, thus FPGA memory can be saved. The W(k) module calculates (6) in integer pipeline (Fig. 4). Sum of samples in every *N*-point window (*sum(k)*) is passed to the LCM(k) module for calculation of (3). Correlation results from the filter output (*ccorr*), *LCM* results and *W* results are written to the memory buffers in the normalization module. These buffers synchronize input streams. Synchronized streams from buffers outputs are processed in floating point NCC(k) pipeline. Precalculated sums of squares (*PS*) of templates are stored in RAM as floating point numbers. Next subsections describe components developed for the *ncctop*.

A. Correlation filters

Correlation filter calculates cross-correlation (2) between samples window and n_s templates. New sample is entered into

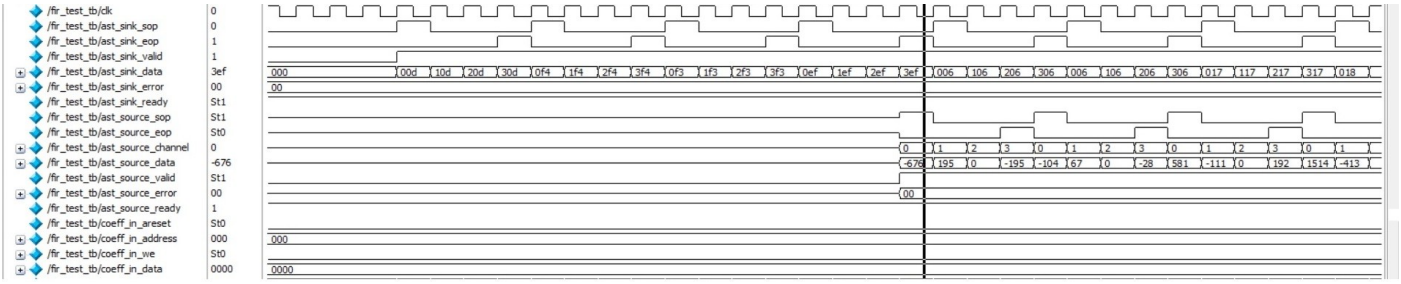

Fig. 2. Timing diagram from a simulation of the FIR II correlation filter.

the filter input every S_t*n_s or more clock cycles and produces n_s L(k) values on output. Filters described in this section support setting the same S_t value for every template. The NCC_FIP is the filter implementation based on an instance of the FIR II IP core from Quartus Prime [17]. Coefficients are values of templates *s(i)* in U2 code. Input and output interfaces of the filter are based on the Avalon-ST interface. Filter coefficients can be reloaded in run-time using separate interface. FIR II IP core optimizes hardware utilization by applying time-division multiplexing. The time-division multiplexing ratio *TDM* is defined as $TDM = f_{clk} / f_{smp}$, where f_{clk} is clock rate, f_{smp} is the sample rate of input channels (*InputChannelNum)*.

Filter size *N*, *TDM*, number of filter banks and others parameters are configured using parameter editor. For example, if $f_{clk} = f_{smp} = 200$ MHz and *InputChannelNum* = 1, then the filter will process single stream of data. Filter computational throughput is 1 result per 1 clock cycle. Filter configuration f_{clk} = 200 MHz, f_{smp} = 50 MHz, *InputChannelNum* = 4 and four coefficient banks can be used for calculation of cross-correlation in single data stream with four templates. Figure 2 shows timing diagram for this configuration and 8-bit samples. Samples are entered into *ast_sink_data[7:0]* input every 4 clock cycles, beginning from the cycle when *ast_sink_sop* is set to 1. Coefficient banks with templates data are switched in sequence 0, 1, 2, 3 by setting *ast_sink_data[9:8]* input. Correlation results are available on *ast_source_data* output in groups of four values, one value for each template. For this example, the latency is 15 clock cycles.

The NCC_MEM (Fig. 3) is a programmable filter based on FPGA memory blocks. This filter supports larger values of *TDM* and larger values of filter windows. It consists of *CC_PE* processing elements (*PE*), each of them embeds the memory for input samples (*IMEM*), the memory for template samples (*CMEM*) and the multiplication module. The NCC_MEM is parametrized in word and address lengths, *CC_PE*, selection between RAM-based or register-based memories, selection between single or dual port RAM modules, maximum number of templates 1 – 256 *(CC_NSIG*), maximum value of S_t and component latencies. *N*, n_s and S_t are programmable by writing to control registers located in external interface module. *N* can be set up to 65536, n_s = 1,…, 128, S_t = 1, …, ceil(*N* / *CC_PE*). Cross-correlation is calculated with templates of size *CC_NPE* * S_t samples. If template size is not integer multiple of *CC_NPE*, then template should be extended by zero padding. The filter throughput is S_t clock cycles per template, S_t*n_s clock cycles per input sample. Before starting a stream processing, registers and memories are reset. Template samples are written to template memories. After initiation, input samples are written to the IMEM0 every S_t*n_s or more clock cycles. In Fig. 3.a), the architecture with single port memories is depicted. RAMs are implemented by using the *altsyncram* LPM. RAM output is unregistered. Read from an address being simultaneously written provides new data written. Samples read from IMEM block are written to the next block and entered into the first inputs of multipliers. Template samples read from CMEM are entered into the second inputs of multipliers. Multiplication results are summed in parallel adder.

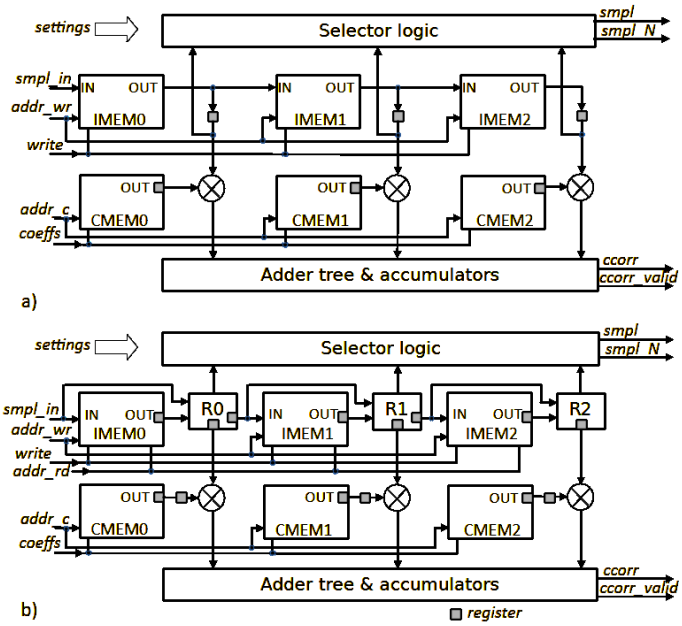

Fig. 3. Architectures of the NCC_MEM filter, CC_PE = 3. a) with single port memories; b) with dual-port IMEMs.

Output stage consists of CC_NSIG accumulators, each of them for one template. Control unit is built using simple counters. The counter generating address for input memories is set to 0 when S_t = 1. For S_t > 1, the IMEM address changes every clock cycle in sequence {0,1,…,S_t-1},{S_t-1,0,1,…,S_t-2}, {S_t-2,S_t-1,0,1,…,S_t-3} ... Every sequence in the curly bracket is repeated n_s times. CMEM address is changing every clock cycle in sequence {0,1,…,S_t*n_s-1,0,1,…}. For this data flow, each of PE multiplies S_t adjacent samples by corresponding to them template samples. In this architecture, there are direct unregistered RAM to RAM connections. These connections may have influence on clock frequency. Fig. 3b) shows architecture containing dual-port RAMs with registered RAM outputs. In this case, the *altsyncram* is configured as *RAM:2-*

port with single clock with registers on input and output. Separate ports are used to write and read. Data read from a RAM block is latched in register block (R0 – R2) before writing to then next RAM block. Register block embeds single register for storage data from output of the previous memory block if $S_t > 1$ or from output of previous register block if $S_t = 1$. Write address is changing in sequence {0, 1,…, S_t-1, 0, 1, ...}. Read address is changing every clock cycle in sequence {0, S_t-1, S_t-2,…,1}, {1, 0, S_t-1, …, 2} ... Every sequence in the curly bracket is repeated n_s times. Sample frequency can have any value not larger than $f_{clk}/(S_t*n_s)$.

Multiplication and addition are configured for U2 integer data. Word-length of filter results is set to CC_COEFF_N + CC_IN_N + log2(*filter_size*), where CC_COEFF_N is template bit-width, CC_IN_N is input bit-width, *filter_size* is the size of filter window. For multiplication, instances of LPM_MULT IP with pipeline are embed, additions are described using Verilog + operator. Reduction of word-length is supported by setting two scaling constants. The one constant determines number of LSB bits discarded from multiplication result, the second constant determines number of LSB bits discarded from filter result.

B. The normalization modules

Integer pipeline for the W(k) calculation is shown in Fig. 4. The running sum and the running sum of squares in every N-sample window are calculated in integer accumulators. Accumulators calculate $sum(a) = sum(a - 1) + f(a) – f(a - N)$ and $sum2(a) = sum2(a - 1) + f^2(a) – f^2(a - N)$.

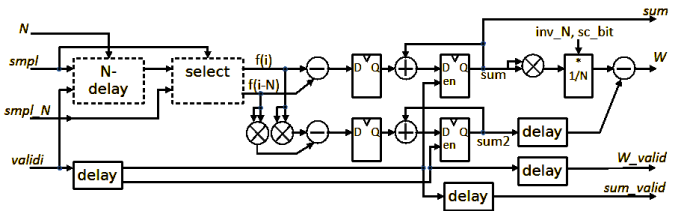

Fig. 4. Architecture of the W(k) module.

Instances of the LPM_MULT library module are applied for multiplications, sum^2/N is implemented as fixed point multiplication by scaled value of *1/N*. Number of bits in *1/N* mantissa, number of fractional bits in *W(k)*, multipliers configurations (pipeline depth, DSP/LUT) are configurable. N-delay memory delays incoming samples by *N* sampling periods. Delayed signal from the output is used as an outcoming sample *f(i - N)*. The N-delay memory is optional and should be included for the NCC_FIP. For the NCC_MEM, the *f(i - N)* sample can be selected from the *sampl_N* output of the correlation filter. Architecture from Fig, 4 can be straightforwardly extended to support processing of multiple templates which sizes can differ. Two extensions should be made. The first is adding two-stage delay memory instead of the *N-delay* and the *select*. The first stage can contain a memory buffer or select the *smpl_N* to fetch the sample $f(i - N_{min})$, where N_{min} is the minimum template size from processed group. The second stage will contain additional n_s - 1 delay buffers. Each of them adds the delay equals to $N_{dt} = N_t - N_{min}$, N_t is the size of *t*-th template. Consequently, the last sample from the window ($f(i - N_t)$) for currently processed template can be selected from the output of an appropriate delay branch. The second extension is adding two register banks (or two accumulator banks) instead of two single registers in accumulators. Each bank contains a register storing results for every N_t window. Finally, selector logic chooses *sum* and *sum2* from output of an appropriate register.

LCM module consists of 32-bit floating point multiplier. Template means, prior written to RAM, are multiplied by the running sums. The normalization module (Fig. 5) includes 32-bit floating point instances: two multipliers, one subtractor, one divider and one square root. The arithmetic modules are configured in the 32-bit IEEE-754 standard (FP32). Square root can be also substituted by look-up table after input conversion to the range of -1 ,.., 1. Verilog codes of floating point modules were generated using IP Catalog tool from Quartus. The ALTERA_FP_FUNCTIONS library was chosen to generate arithmetic and format conversion instances. IP generator enables customization for FP32, double precision or custom precision formats. The latency of generated instance depends on data formats and target clock frequency. Target clock frequency was set to 200 MHz.

L(k) data are written to the FIFO synchronization buffer in the BUF1 when its valid signal (*L_valid*) is set active, LCM(k) data are written to the second FIFO in BUF1 when *LCM_valid* is set active. L and LCM data are synchronously read from FIFOs, $NUM^2(k) = (L(k) - LCM(k))^2$ is calculated and written to the FIFO in the BUF2. W(k) data after conversion to FP32 are multiplied by PS(k) for every template in the group. Next, the denominator data are written to the FIFO in the BUF2. Control logic generates signals for BUF2 writing, address for PS RAM and *NCC_valid*. Numerator and denominator data are synchronously read from BUF2, next NCC(k) = sqrt(Numerator^2(k) / Denominator(k)) is calculated for every template.

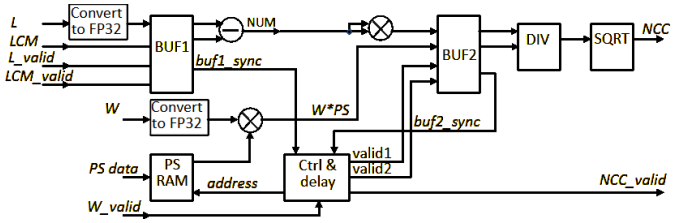

Fig. 5. Structure of the normalization module

C. Implementation results

Tables I and II show implementation results of the *ncctop* for Cyclone V 5CGTFD9D5F27C7 FPGA. Intel Quartus Prime Lite was used for synthesis, implementation and timing analysis. L(k) was calculated with full precision, number of mantissa bits in *W(k)* module was set to 18. PE is the number of processing elements in the filter, N_{max} is the maximum value of *N*, *N* has the same value for every template, N_s is the maximum number of templates.

The advantages of the NCC_FIP are lower ALM and memory usage and higher clock frequency. Increasing of TDM slightly increases ALMs allocation. The NCC_MEM enables programming of *N* and n_s but requires sufficient amount of FPGA memory. M10K memory blocks in Cyclone V FPGA can be configured for several depth - width values, among them are:

1k x 8 bit, 512 x 16 bit, 256 x 32 bit [18]. When depth of RAM blocks is lower than the technological value, the M10K allocation can increase. Memory optimizations for save FPGA memory blocks, e.g., merging blocks, can be further optimizations of this architecture.

Calculation throughput for single template from the group is 1 NCC per *ceil(N/PE)* * n_s clock cycles for NCC_MEM; 1 NCC per TDM clock cycles for NCC_FIP. Calculation latency is less than 70 clock cycles. Table II summarizes clock performance for configurations from Table I. The f_{clkmin} is the clock frequency of slow chip corner, the f_{clkmax} is the clock frequency of fast chip corner [19]. Results of NCC_MEM Single-port RAM are estimated for the same configurations as for the NCC_MEM Dual-port RAM.

TABLE I
EXPERIMENTAL RESULTS OF NCCTOP. 8-BIT SAMPLES, 8-BIT TEMPLATES.

PARAMETERS			AREA			
NCC_FIP						
N	TDM	n_s	ALMs	Registers	M10K blocks	DSP
64	1	1	2124 (2%)	5169	16	42
	4	4	3354 (3%)	6288	16	42
128	1	1	2423 (2%)	5897	16	74
	4	4	4828 (4%)	8138	16	74
512	1	1	10202 (9%)	23200	16	222
	4	4	14414 (13%)	20039	16	266
1024	1	1	43190 (38%)	88213	17	222
	4	4	40158 (36%)	69384	17	342
NCC_MEM						
PE	N_{max}	N_s	ALMs	Registers	M10K blocks	DSP
Dual-port RAM (Fig. 3 b))						
64	8192	4	3782 (3%)	9403	92	74
		32	3866 (4%)	9430	284	74
128	16384	4	5688 (5%)	13597	169	138
		32	5730 (5%)	13624	553	138
512	8192	32	16598 (15%)	38271	630	342
	65536	4	16789 (15%)	38480	630	342
Single-port RAM (Fig. 3 a))						
512	65536	4	12859 (11%)	30216	629	342

TABLE II
EXPERIMENTAL RESULTS OF CLOCK PERFORMANCE

Architecture	f_{clkmin} [MHz]	f_{clkmax} [MHz]
NCC_FIP	160 - 205	260 - 370
NCC_MEM Dual-port RAM	160 - 200	240 - 310
NCC_MEM Single-port RAM	150 - 200	230 - 310

Experiments showed that increasing of the ALM allocation may lead to decreasing of the maximum value of clock frequency. The lowest frequency values were reported for the largest architectures NCC_FIP N = 1024 TDM = 1, NCC_MEM PE = 512.

IV. The NCC_FD Design Based on FFT

The NCC_fd design includes a correlation filter designed in the frequency domain:

$$corr(k) = \sum_{i=0}^{N-1} f(i+k)(s(i) - \mu_s) \quad <=>$$
$$FFT^{-1}(FFT(f_k) * conj(FFT(s - \mu_s))) \quad <=> \quad (7)$$
$$FFT^{-1}(FFT(f_k) * FFT(s_{rev} - \mu_s))$$

where

conj is complex conjugate, s_{rev} is s in the reverse order.

FFT-based correlation is computed using overlap-save algorithm [2] applied to stream processing. Stream processing steps are as follows.

1. Pre-processing. Templates are normalized by subtracting means, next zeros are padded to the length of FFT. FFT is calculated for every pre-processed template. FFT coefficients are rounded to the chosen format. As a result, FFT_M_t, $t = 0, \ldots, n_s - 1$, coefficients are written to memory buffer.
2. Input samples are divided into blocks of the size $L\_size = FFT\_size - M + 1$, where M is the length of the longest template in the set. Last M - 1 samples from previous block are the first samples in the next block, followed by L_size new samples. Before the first block, M - 1 zeros are inserted at the beginning.
3. FFT is calculated for the input block (*FFTB*).
4. Complex multiplication of FFTs:
$MUL_{Bt} = FFTB * conj(FFT\_M_t)$.
5. Inverse FFT: $IFFT_{Bt} = FFT^{-1}(MUL_{Bt})$. First L_size values of $IFFT_{Bt}$ real component are L(k) data. Remaining $M - 1$ values of IFFT are discarded.

If lengths of templates differ, first valid correlation value is shifted by $M - N_t$ positions, where N_t is the length of t-th template.

Steps 3 - 5 are repeated for every template.

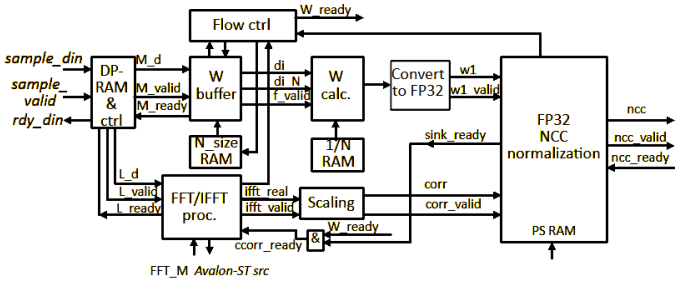

Fig. 6. Block diagram of the *ncc_fd_proc*

Block diagram of the main module of the NCC_fd, denoted as *ncc_fd_proc*, is shown in Fig. 6. After reset prior to writing input samples, M - 1 leading zeros are written to the first port of dual-port DP-RAM. The size of DP-RAM is 2**FFT_size* samples. Input data are read from the second port of the DP-RAM in blocks of FFT_size samples. Data streams for FFT (L_d, L_valid) and for W(k) (M_d, M_valid) are formed. Each of sample blocks is repeated n_s times. Next, read counter is decremented by M - 1 to the address of first sample in next

block. The *W buffer* consists of one FIFO buffer and one two-stage delay buffer. The FIFO buffer stores samples for the time related to FFT to IFFT latency. The delay buffer reads incoming sample (*di*), an outcoming (*di_N*) samples used for calculation of the W(k). Sizes of templates in the processed group can be different and they are programmed in run-time. The flow control ensures proper synchronization between the W(k) stream and the *ifft_real* stream. Correlation results after scaling of the IFFT real data are written to the synchronization buffer in the *NCC normalization* module. W(k) data, calculated in a similar way as is described in IIIB, are written to the second synchronization buffer. The *NCC normalization* module includes FP32 multiplication, division, and square root IPs from the ALTERA_FP_FUNCTIONS library. Finally, the NCC(k) stream for n_s templates is available on the module output.

FFT/IFFT based correlation module (Fig. 7) consists of two instances of FFT IP from the Quartus Prime IP library [16]. Data flow between modules is based on the Avalon-ST source (*src*) and sink (*snk*) interfaces. To save FPGA ALM resources, integer FFT has been used. Output bit order for this core configuration is set to radix-2. Input bit order of FP32 IFFT must be set to radix-4. Therefore, reorder buffer (*RBuffer*) between FFT output and complex multiplier is required to change the order to radix-4. The complex multiplier is implemented using the ALTMULT_COMPLEX IP. If FP32 FFT module is used, the reorder buffer is skipped.

The *ncc_fd_proc* throughput is *L_size* NCC results per *FFT_size* clock cycles per single template. The latency is $(2+L_{reo})*FFT\_size + L_{fl}$ clock cycles, where

L_{reo} is 1 if reorder buffer is used, 0 if reorder buffer is skipped,

L_{fl} is the latency added by other modules in the data path.

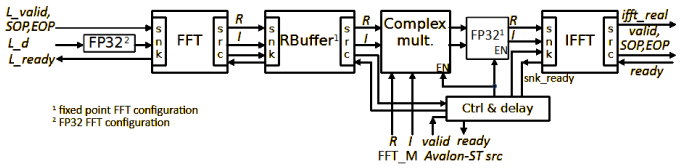

Fig. 7. The architecture of the FFT/IFFT module.

Example of a timing diagram from simulation of the *ncc_fd_proc* is shown in Fig. 8. In this stimulus, there are four templates with sizes of 128, 104, 140, and 160 samples, input data block has 2000 samples, 1024-point FFT is used. Overlap size is set to $M = 159$. Input data are divided into three blocks. The *ncc_fd_proc* calculates L_size = 865 NCCs in 1024 clock cycles. Every input block is repeated four times, one block per one template. As it is shown, the IP core enables connection to a busy DMA channel using Avalon-ST flow control. This feature was simulated by randomly setting the *ncc_ready* input to 0 during processing of blocks no. 2 and no. 3.

TABLE III
EXPERIMENTAL RESULTS OF NCC_FD_PROC OBTAINED FOR: 8-BIT SAMPLES, 8-BIT TEMPLATES, INTEGER FFT, INTEGER COMPLEX MULTIPLICATION, FP32 IFFT. THE *FFT_size* IS PROGRAMMABLE

FFT/IFFT length	N_s	ALMs	Registers	M10K blocks	DSP	$f_{clkmin} - f_{clkmax}$ [MHz]
1024	16	15245 (13%)	29058	146	67	140 - 230
4096	16	18759 (17%)	34861	278	81	135 - 230
16384	16	22534 (20%)	40620	733	96	135 - 220

Preliminary results of *ncc_ft_proc* implementation are shown in Table III. Memory allocation and latency can be further reduced after applying FFT and IFFT with compatible bit orders.

V. DESIGN VERIFICATION AND RESULTS

Described designs were verified in ModelSim simulations and prototyped on hardware. Reference program (RP) was written in Matlab to generate files used in testbenches. Matlab *xcorr* and *corrcoef* functions were used for calculation of cross-correlation and NCC. RP generated synthetic signals as templates. Templates after multiplying by attenuation factors were inserted into the input waveform and noise was added. Cross-correlation and NCC results from RP were compared sample-by-sample with results from the simulation. Fig. 9 shows an example stimulus. The input signal is a sum of

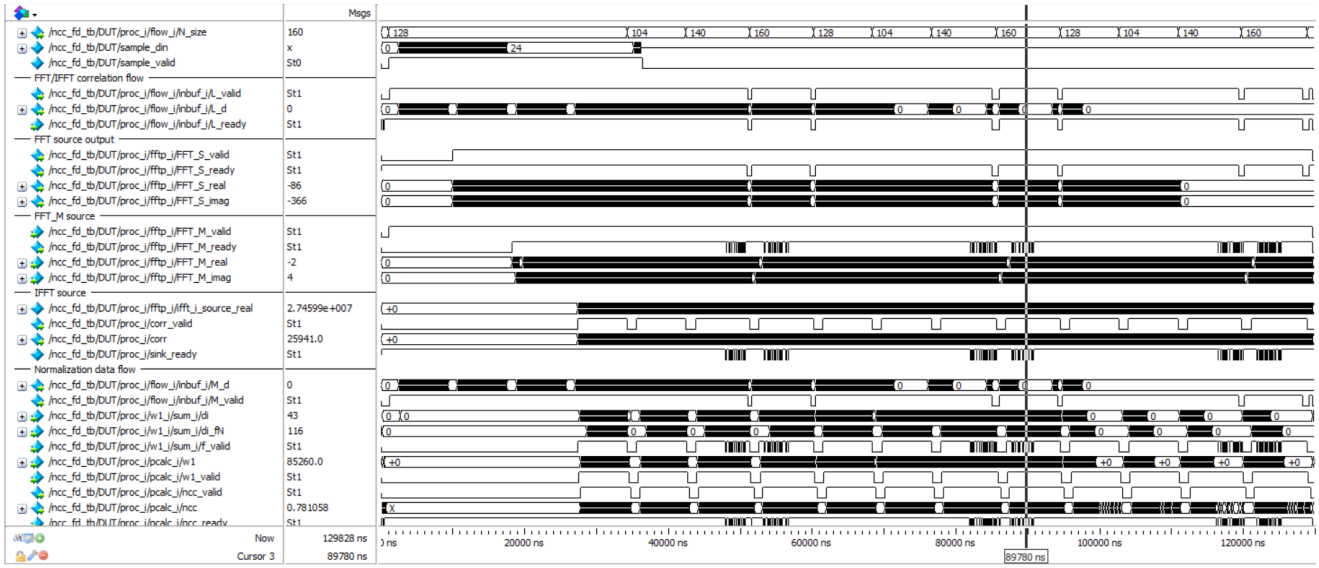

Fig. 8. Timing diagram from the simulation of the *ncc_fd_proc*.

a slowly changed component, DC, noise and four attenuated templates. The first template is a sine wave, the second is a frequency modulated sine wave, the third is a rectangle pulse, the fourth is a Dirichlet pulse. The length of each of template is 300 samples.

TABLE IV
REPORTED ACCURACY

Δ_{NCCmax}	$\lvert\delta_{NCCmax}\rvert$[%]	$\lvert\Delta_{NCC}\rvert$
NCC_td		
-2.0e-06	2.8e-04	2.0e-06
NCC_fd integer FFT, FP32 IFFT		
1.3e-04	0.018	2.3e-03
NCC_fd integer FFT, FP32 IFFT, 24-bit IFFT input data		
-4.2e-04	0.046	5.6e-02

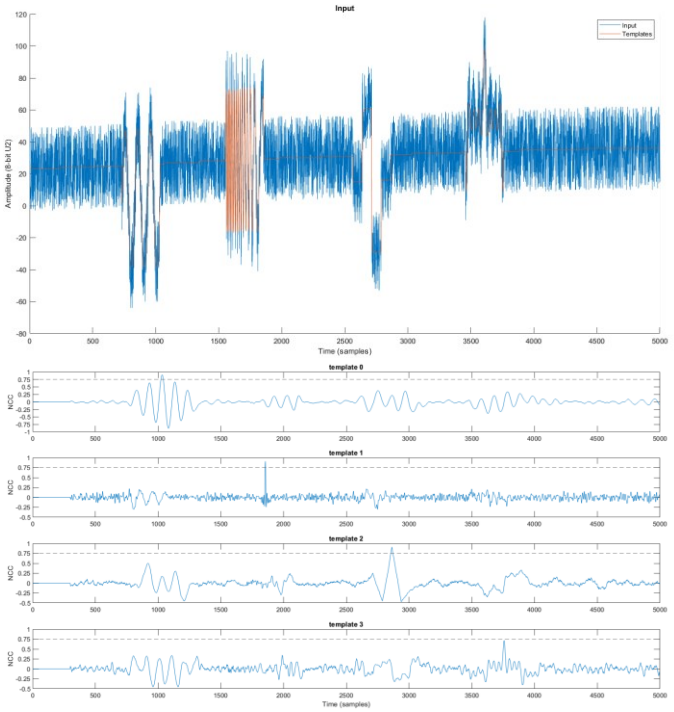

Fig. 9. Time diagram of an example stimulus (upper picture). Lower pictures show the Pearson coefficients calculated between this stimulus and four templates.

Reported accuracy for this stimulus is shown in Table 4. Δ_{NCCmax} is the maximum absolute error for maximum NCC values, $\lvert\delta_{NCCmax}\rvert$ is the maximum relative error in % for maximum NCC values, $\lvert\Delta_{NCC}\rvert$ is the maximum absolute error in the whole timeseries. Sources of rounding errors in NCC_td are *1/N* multiplying and floating point operations. NCC_fd accuracy has been lower what is related to applying the integer FFT. Reducing word-length on IFFT inputs to 24-bits by discarding 14 least significant bits of complex multiplication results has yielded an accuracy of 0.046 %.

Table V shows the comparison between the proposed NCC_FIP/NCC_MEM architectures and [5]. Proposed designs enable implementation of NCC function with small FPGA resources. Additionally, the NCC_MEM adds programming in run-time.

NCC_fd configured with 16384-point and 4096-point FFT/IFFT may provide better throughput for templates with sizes up to thousands of samples and it can be implemented on selected FPGA chip. One of the main differences between NCC_td and NCC_fd is significantly larger latency of NCC_fd.

TABLE V
DESIGNS COMPARISION
FIP1: NCC_FIP, N=64, TDM=1, n_s = 1, 8 x 8 bit
FIP2: NCC_FIP, N=64, TDM=1, n_s = 1, 12 x 12 bit
MEM1: NCC_MEM, PE=64, N_{max}=8192, N_s = 4, 8 x 8 bit
MEM2: NCC_MEM, PE=64, N_{max}=8192, N_s = 4, 12 x 12 bit

	[5] PCI	FIP1	FIP2	MEM1	MEM2
Template size (N)	64	2,…,64	2,…64	2,…,8192	2,…,8192
Throughput (NCC/clk)	1	1	1	1 - 128	1 - 128
Xilinx Artix-7 LUT	21349	-	-	-	-
Altera Cyclone V ALM	-	2124	2339	3782	4751
Registers	21524	5159	5876	9403	11715
Memory blocks	0	16	16	92	98
DSP blocks	120	42	42	74	74
Max. frequency [MHz]	137.8	200 - 310	200 - 310	200 - 300	180 - 300

The architecture of the prototype system is depicted in Fig. 10. Tests were performed using Terasic OVSK board, containing Cyclone V 5CGTFD9D5F27C7 FPGA. Altera PCIe Windows x64 driver and TERASIC_PCIE_AVMM.dll were used as an interface between user software and FPGA hardware, including write and read DDR memory. Fully functional NCC module consists of the computation core (the *ncctop* or the *ncc_fd_proc*) and the interface logic with internal registers. The NCC module has been connected to the PCIe interface. 32-bit Avalon-MM port *av* writes and reads to/from internal registers. 128-bit Avalon-ST port *avst_in* writes input samples from DDR3 to the module using DMA. Additional DMA channel is used to write *FFT_M* data from the DDR3 to the NCC_fd. 128-bit Avalon-ST port *st_out* writes calculated NCC stream to DDR3 using separate DMA channel.

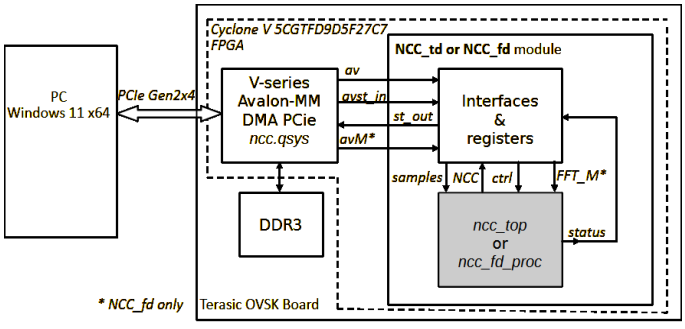

Fig. 10. Hardware for prototyping of the NCC modules

SoPC subsystem *ncc.qsys* was designed using the Platform Designer from Quartus. The *ncc.qsys* contains the *V-series Avalon-MM DMA PCIe* and the *mSGDMA* IP cores [15]. Test programs were written in C. Results from RP were compared with NCC values from hardware. Performed tests showed that the developed NCC modules work correctly.

VI. CONCLUSION

This paper presents the FPGA IP cores designed for calculation of the Pearson correlation coefficient for signal template matching. NCC functional modules with cross-correlation filtering in the time domain and in the frequency domain followed by NCC normalization were described. In this work, parametrizable Verilog cores were developed what will simplify the development of a custom FPGA-based hardware. Detailed descriptions of the developed modules together with estimation of FPGA resource usage are presented. NCC_td enables choosing between resource-optimized FIR IP architecture and programmable filter. NCC_fd enables higher throughput for large template sizes up to thousands of samples, with resource requirements which are dependent on FFT/IFFT architectures. Further optimizations of the designs will be targeted to improve performance and add configurations for multi-channel correlation processor.

REFERENCES

[1] J. L. Rodgers, W. A. Nicewander, “Thirteen Ways to Look at the Correlation Coefficient”, American statistician (1988), 42, pp. 59–66, 1988, https://doi.org/10.2307/2685263

[2] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, “Numerical Recipes in C, 2nd Ed.”, Cambridge University Press, 1992.

[3] Correlation coefficients corrcoef, Matlab Help, https://www.mathworks.com/help/matlab/ref/corrcoef.html

[4] J. P. Lewis, “Fast Normalized Cross-Correlation”, In: Vision Interface, pp. 120–123, 1995

[5] A. Cicuttin, I. R. Morales, M. L. Crespo, S. Carrato, L. G. García, R. S. Molina, B. Valinoti, J. F. Kamdem, “A Simplified Correlation Index for Fast Real-Time Pulse Shape Recognition”, Sensors 2022, 22(20), 7697, https://doi.org/10.3390/s22207697

[6] C.H. Moore, W. Lin, “FPGA Correlator for Applications in Embedded Smart Devices”, Biosensors 2022, 12(4), 236; https://doi.org/10.3390/bios12040236

[7] C. La, M. J. Liu, X. F. Li, “The FPGA Implementation of Matched Correlation Filter”, in Proc. 2011 International Conference on Electronics, Communications and Control (ICECC),pp. 1114-1117, 2011, https://doi.org/10.1109/ICECC.2011.6066596

[8] F. J. Iniguez-Lomeli, S. Renaud, J. H. Barron-Zambrano, “A Real Time FPGA-Based Implementation for Detection and Sorting of Bio-signals, Neural Computing and Applications 33(22)”, pp. 12121–12140, 2021, https://doi.org/10.1007/s00521-021-05853-7

[9] S. Adrián-Martínez, M. Ardid, M. Bou-Cabo, I. Felis, C. D. Llorens, J.A. Martínez-Mora, M. Saldaña, “Acoustic Signal Detection Through the Cross-Correlation Method in Experiments with Different Signal to Noise Ratio and Reverberation Conditions”. In: Garcia Pineda, M., Lloret, J., Papavassiliou, S., Ruehrup, S., Westphall, C. (eds) Ad-hoc Networks and Wireless. ADHOC-NOW 2014. Lecture Notes in Computer Science(), vol 8629. Springer, Berlin, Heidelberg, pp. 66-79 ,2015. https://doi.org/10.1007/978-3-662-46338-3_7

[10] D. Lee, S. Lee, S. Oh, D. Park, ”Energy-Efficient FPGA Accelerator with Fidelity-Controllable Sliding-Region Signal Processing Unit for Abnormal ECG Diagnosis on IoT Edge Devices”, IEEE Access, Volume 9, pp. 122789-122800, 2021, https://doi.org/10.1109/ACCESS.2021.3109875

[11] Y. Huang, H. Bao, X. Qi, “Seismic Random Noise Attenuation Method Based on Variational Mode Decomposition and Correlation Coefficients”, Electronics 2018, 7(11), 280, https://doi.org/10.3390/electronics7110280

[12] M. Faisal, R. T. Schiffer, M. Flaska, S. A. Pozzi, D. D. Wentzloff, “A correlation-based pulse detection technique for gamma-ray/neutron detectors”, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, Volume 652, Issue 1, pp. 479-482, 2011, https://doi.org/10.1016/j.nima.2010.10.072

[13] M. Garrido, J. Grajal, M. A. Sanchez, O. Gustafsson, "Pipelined Radix-2^k Feedforward FFT Architectures", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 21, no. 1, pp. 23-32, Jan. 2013, https://doi.org/10.1109/TVLSI.2011.2178275

[14] M. Chiosa, T. B. Preußer, M. Blott, G. Alonso, “AMNES: Accelerating the computation of data correlation using FPGAs”, in Proc. of the VLDB Endowment, Volume 16, Issue 13, pp. 4174-7187, 2023, https://doi.org/10.14778/3625054.3625056

[15] K. Mroczek, “SoPC-based DMA for PCI Express DAQ Cards”, International Journal of Electronics and Telecommunications (IJET), Vol. 67 No. 4, pp. 565-570, 2021, https://doi.org/10.24425/ijet.2021.137847

[16] Intel® Corporation, FFT IP Core User Guide, Intel Quartus Prime Design Suite, v. 17.1

[17] Intel® Corporation, FIR II IP Core User Guide, Intel Quartus Prime Design Suite, v. 17.1

[18] Intel® Corporation, Cyclone V Device Handbook, Volume 1: Device Interfaces and Integration

[19] Intel® Quartus® Prime Standard Edition User Guide Timing Analyzer, v. 18.1