Research Article

A Low gate count reconfigurable architecture for biomedical signal processing applications

Nupur Jain[1] · Biswajit Mishra[1] · Peter Wilson[2]

Received: 25 September 2020 / Accepted: 22 February 2021 / Published online: 8 March 2021

Abstract
A new reconfigurable architecture for biomedical applications is presented in this paper. The architecture targets frequently encountered functions in biomedical signal processing algorithms thereby replacing multiple dedicated accelerators and reports low gate count. An optimized implementation is achieved by mapping methodologies to functions and limiting the required memory leading directly to an overall minimization of gate count. The proposed architecture has a simple configuration scheme with special provision for handling feedback. The effectiveness of the architecture is demonstrated on an FPGA to show implementation schemes for multiple DSP functions. The architecture has gate count of ≈25k and an operating frequency of 46.9 MHz.

Keywords Biomedical signal processing · Reconfigurable architectures · Low gate count architecture · Digital signal processing

1 Introduction

Advances in Integrated Circuit (IC) technology have resulted in miniaturized devices useful for wearable biomedical applications. Custom and configurable solutions for Electrocardiogram (ECG) processing have been discussed extensively in the literature [1–4]. As an example, [5] discusses a low-energy microprocessor (μP) for R-wave detection, heart rate calculation and arrhythmia based on the R-R interval with a 78% detection accuracy. The use of microprocessors provides flexibility in configuration that translates to cost-effective and time-efficient solutions but adversely affects the power budget for compute-intensive operations. Researchers in [6] discuss low power system that combines off-the-shelf μP with dedicated ECG processing for data compression and arrhythmia detection. In [7, 8], custom accelerators for DSP functions used in biomedical signal processing are discussed. These accelerators in conjunction with a customized low power μP are used for ExG (EEG (Electroencephalogram), ECG) processing, targeting a large subset of biomedical applications. A miniaturized system in [9] demonstrates a limit on radio transmissions indicating alerts and alarms instead of continuous streaming of raw data while monitoring critical physiological signals in patients. It provided a significant increase in the power efficiency of the overall system due to a decrease in radio utilization, emphasizing the need to maximize local processing. Generic hardware accelerators discussed in [8, 10, 11], target common signal processing techniques predominantly used in biomedical applications. This ensures the use of accelerators in multiple varied algorithms and provides an energy-efficient solution with the flexibility to map a large set of biomedical applications. However, multiple accelerators can increase the overall gate count of the system. The potential performance benefits of a low gate count architecture are discussed in [12] and an analysis of the effect of gate count

✉ Biswajit Mishra, biswajit_mishra@daiict.ac.in; Peter Wilson, P.R.Wilson@bath.ac.uk | [1]VLSI and Embedded Systems Group, DA-IICT, Gandhinagar 382007, India. [2]Dept of Electronic and Electrical Engineering, University of Bath, Bath BA2 7AY, United Kingdom.

and critical path of architecture in the sub-threshold region on power consumption is highlighted in [13].

It is possible to combine the performance and efficiency benefits of hardware with the flexibility of software by using reconfigurable systems where the hardware (re) configures itself to perform multiple functions of an application in a time-multiplexed manner. The regularity of certain functions used in biomedical signal processing algorithms is exploited in this work which results in reduced overhead and consequently, a lightweight architecture is obtained. A novel reconfigurable architecture based on shift-accumulate operation is configurable for frequently processed algorithms. The architecture has a configurable datapath and supports a range of biomedical signal processing algorithms and is verified on a field programmable gate array (FPGA) platform.

The developed architecture is implemented on an FPGA as a proof-of-concept and can be realized as an ASIC acting as an accelerator for compute-intensive operations alongside an embedded processor. The compute-intensive operations, such as CORDIC, DCT, *etc.*, are often seen in biomedical signal processing algorithms and take prolonged time to process such operations on embedded processors. A hardware design with dataflow-based execution offers processing in reduced number of clock cycles as compared to an embedded processor. Additionally, the developed architecture is configurable and thus can span a large set of biomedical algorithms imitating (to some extent) the complete programmability advantage of embedded processors.

The types of biomedical functions addressed are presented in Sect. 2. The overview of the proposed architecture is discussed in Sect. 3 followed by different architectural topologies in Sect. 4. The mapping methodologies for target functions are discussed in Sect. 5. The results for different mapping cases on the hardware are presented in Sect. 6. Conclusion and future work are discussed in Sect. 7.

2 Functionality profile of biomedical applications

Table 1 presents the overview of common DSP functions used in biomedical applications. The majority of the tabulated applications report the frequent use of following functions FIR and IIR Filters, Differentiation, Moving average, maxima and minima, FFT, DWT, DCT and complex trigonometric functions. Finite Impulse Response or Infinite Impulse Response (FIR or IIR) filtering is used primarily for noise removal and feature extraction. The differentiation function is used to track the variation pattern of a signal by means of slope information. Moving average, median and maxima functions are used to extract the peak information of a signal. The Fast Fourier Transform (FFT) and Discrete Wavelet Transform (DWT) are used for analysis of signals in the frequency domain. The classical approach of estimating blood saturation involves the Discrete Cosine Transform (DCT) computations [14]. Additionally, the DWT and DCT algorithms are reported to be used for the signal compression purpose to reduce the transmission share of the radio [6]. Finally, complex trigonometric function computations are reported alongside the fundamental operations of multiplication, addition, scaling, division, *etc.* [15, 16].

Clearly, it can be observed from Table 1 that FIR or IIR, CORDIC, DWT, DCT, differentiation and moving average form the dominant operations encountered in biomedical signal processing applications. As a consequence, these operations constitute the target domain functions for any signal processing hardware platform. In the next section of this paper we show that by manipulating a simple

Table 1 Digital Signal Processing Functions in Ambulatory Biomedical Applications

Signal	Application	DSP functions
ECG	Onset and Duration of QRS [17]	Infinite Impulse Response (IIR), Magnitude of vector ($\sqrt{\cdot}$, $(\cdot)^2$, addition), division, median
	QRS Detection [16]	IIR, Finite Impulse Response (FIR), $\frac{d}{dx}$, $(\cdot)^2$, moving average
	QRS Detection [18]	FIR, $\frac{d}{dx}$, $(\cdot)^2$, weighted moving average
	QRS Detection [19]	Running slope, multiplication, scaling, average, maxima
	ECG fiducial points [20]	Discrete Wavelet Transform (DWT), local maxima modulus
EEG	Epileptic Seizure onset[21]	FIR
	Seizure Detection[22]	DWT
	Automatic recognition of alertness [23]	DWT
PPG	Pulse Oximetry[24]	FIR, log
	Blood Saturation [14]	Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT)
Heart Sound	Auscultation Aid[15]	FFT, tan, sin

mathematical function that forms the basis of these signal processing operations, it is possible to exploit the serial nature of the computation to derive an efficient and reconfigurable circuit.

3 Shift-accumulate (SAC) architecture

3.1 Introduction

As discussed previously in this paper, the shift-accumulate (SAC) architecture is central to potential improvements in the hardware. This section describes the details of the SAC architecture, starting with the mathematical underpinning of the approach.

3.2 Mathematical foundation

In order to establish the mathematical foundation of the SAC approach, an example of a FIR filter can be examined. In an N-tap FIR filter, the output $y[n]$ with inputs $x[n]$ and coefficients b_i is given by:

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + \cdots + b_N \cdot x[n-(N-1)] \quad (1)$$

When the inputs and coefficients are represented as 8-bit binary numbers Eq. 1 can be rewritten as:

$$y[n] = \sum_{i=0}^{N-1} b_i \cdot x[n-i] = \sum_{i=0}^{N-1} \left[\left(\sum_{k=0}^{7} b_{ik} \cdot 2^k \right) x[n-i] \right] \quad (2)$$

(where coefficients (b_i) are expressed in terms of bit value (b_{ik}) and bit weights (2^k))

Rearranging Eq. 2 results in Eq. 3.

$$y[n] = \sum_{k=0}^{7} \left[\left(\sum_{i=0}^{N-1} b_{ik} \cdot x[n-i] \right) 2^k \right] \quad (3)$$

The foundational mathematical equations of the SAC architecture is presented in the paper. The pseudocode of the proposed scheme is shown in Algorithm 1.

The resulting mathematical equation is obtained wherein the partial products can be generated,

Algorithm 1 Pseudo Code for SAC Architecture

```
Load Control Word
Load Coefficients
Load Data
Initialize i

Process:
        For i = 0 to 7
        Multiply coefficient and data serially using bit-wise and operation
        Add partial products
        If Data Valid: Go to Process
End
Log Output Result
```

Table 2 The Comparison of Hardware Implementations of Eqs. 1, 2 and 3 Realizing N-term Multiply-Accumulate

	Eq. 1	Eq. 2	Eq. 3
Nature of Execution	Parallel	Mixed	Serial
ParProd. Generation	64 AND	8 AND	8 AND
ParProd. Accumulation	7 16-b adders	SA	N 9-b adders
Product Accumulation	(N-1) 16-b adders	(N-1) 16-b adders	SA
Equivalent NAND gates	2048N-240	496N-240	143N+240
Gates Savings	13.43×	3.47×	1

AND, OR and XOR gates are accounted as 2, 3 and 4 NAND gates

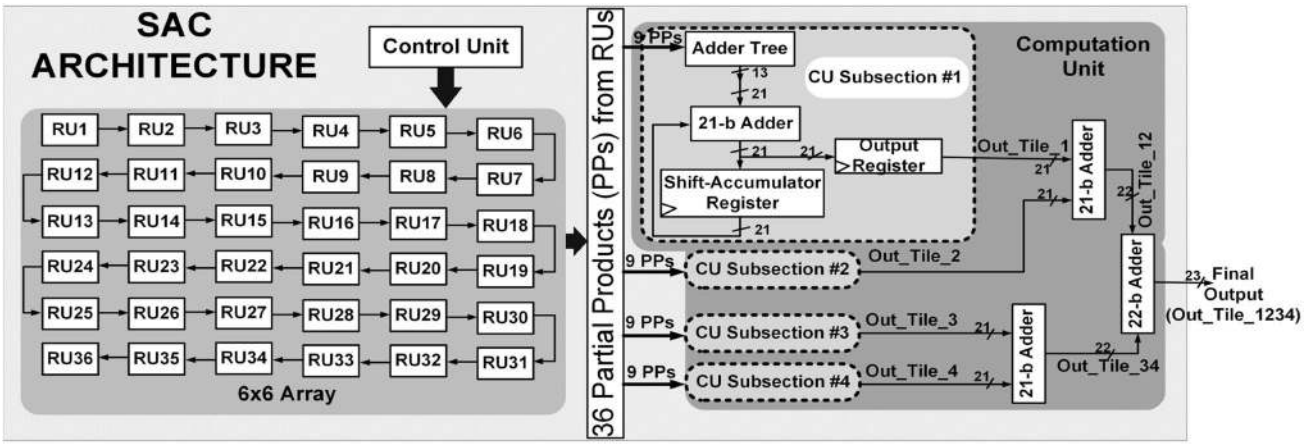

Fig. 1 The Proposed 6x6 Architecture

accumulated sequentially and the common weight partial products then accumulated, showing clearly how the SAC approach can be implemented in this case. The resulting reduction in hardware elements can be clearly seen when the implementation is examined more closely. This function can be realized using a 9-b adder tree containing *N* adders. The final accumulation uses one 16-b SA as opposed to *N* shift-accumulators while implementing hardware using Eq. 2. Eq. 3 implementation uses 143*N*+240 gates with 16 flip-flops for its realization and provides 13.43× and 3.47× saving, respectively, over Eqs. 1 and 2, as indicated in Table 2.

3.3 Architecture and operation

The proposed architecture, shown in Fig. 1, is a Shift-Accumulate (SAC) architecture, consisting of Register Units (RU), a Computation Unit (CU) and a Control Unit. The architecture is arranged as a 6×6 array of functional units called Register Units (RUs) based on Eq. 3. The partial products generated in RUs are fed to the Computation Unit (CU) and the output is obtained after partial products accumulation and shift adjustments. In addition to controlling the function execution, the control unit generates necessary synchronizing signals for RUs and CU.

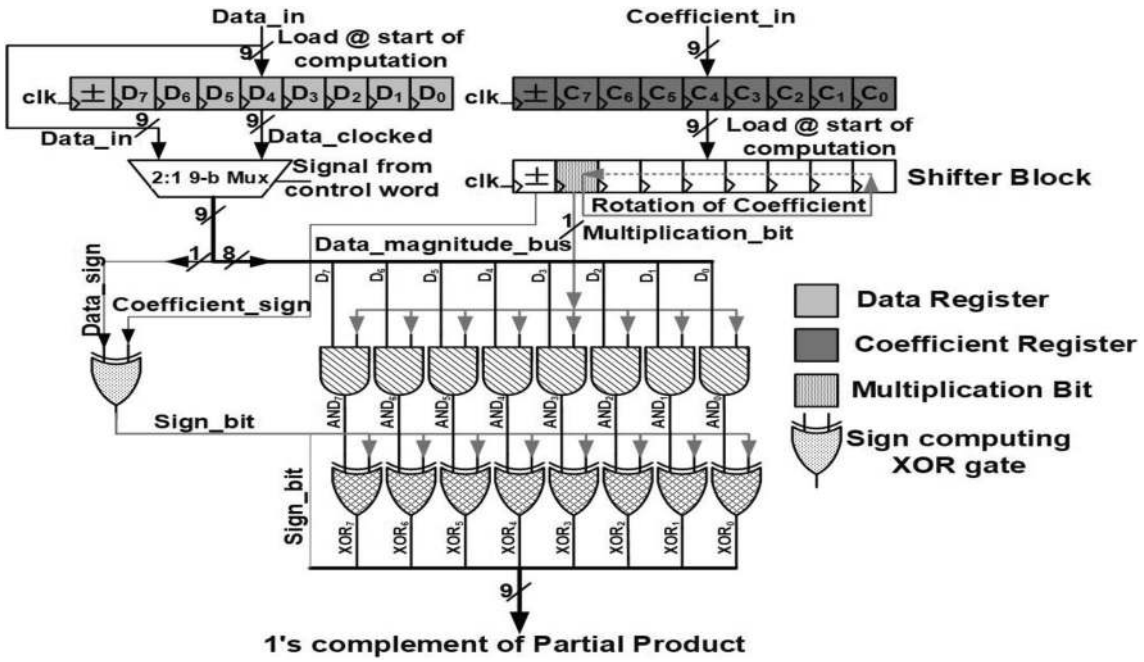

Fig. 2 The Register Unit

3.3.1 Register unit (RU)

The Register Unit consists of registers configured as an 8×1 AND matrix and 8×1 XOR matrix (Fig. 2) to generate partial products. The $x[n]$ input and b_i coefficients are loaded into data register ($D_{7:0}$) and coefficient ($C_{7:0}$) register, respectively. These registers hold the multiplicand and multiplier for the multiplication. The coefficient is further loaded to a shifter block where the multiplier is rotated left by a bit at every clock cycle. The MSB of the shifter is fed to the AND matrix and $D_{7:0}$ follows as the second input. The unsigned partial product thus generated is further converted to a signed partial product by XORing it with the sign bit. A 9-bit 2:1 multiplexer makes the selection between registered and unregistered incoming data. The unregistered data is used in case of functions with feedback where the additional cycles incurred during processing have to be considered for synchronization. Following this, the input data is forwarded to the next RU after eight clock cycles.

3.3.2 Computation unit

As shown in Fig. 1, the Computation Unit (CU) is divided into four identical subsections where each subsection acts on nine partial products. It consists of an adder tree, a shift-accumulator and an output register. The adder tree has ripple carry adders that adds nine partial products each of 9 bits. The 23-bit output register is loaded with the final sum once the serially computed multiplication concludes.

3.3.3 Operation

The coefficient register is loaded with the multiplier in one cycle followed by multiplicand loading in the data register and multiplier forwarded to shifter block in the subsequent cycle (Cycle #1). The multiplicand is ANDed with 7^{th} bit or MSB of multiplier generating the unsigned partial product. In the next clock (Cycle #2), the shifter block would have rotated the multiplier left by one place and 6^{th} bit becomes MSB now as illustrated in Fig. 3. The signed partial products for 6^{th} bit are generated and are added in adder tree. In such a case, the shift-accumulator block has a 7^{th} bit partial product sum (PP_7) that is shifted left by one place (multiplied by 2^1). The 6^{th} partial product sum (PP_6) is added with the shifted PP_7 in the 21-b adder. The shift-accumulator register is now loaded with $PP_7 \cdot 2^1 + PP_6$. In the following clock (Cycle #3) *(due to another left shift) PP_7 is multiplied by 2^2 and PP_6 is multiplied by 2^1. This is added with 5^{th} bit partial product sum that results in $PP_7 \cdot 2^2 + PP_6 \cdot 2^1 + PP_5$. This continues till the shift-accumulator register holds the final output of $PP_7 \cdot 2^7 + PP_6 \cdot 2^6 + PP_5 \cdot 2^5 + PP_4 \cdot 2^4 + PP_3 \cdot 2^3 + PP_2 \cdot 2^2 + PP_1 \cdot 2^1 + PP_0$ after 8 clock cycles.

3.4 Configurable datapath and control word

The proposed Shift-Accumulate (SAC) architecture supports functions that can be expressed as multiply-accumulate operation using a control word. It should be noted that certain functions may not require all the Register Units (RU) and unused RUs can be isolated by means of configurable datapath.

The control word is 54 bit wide and has *Code*, *Config*, *Feedback* and *Resolution* fields as shown in Fig. 4. The 5-bit *Code* field is used to identify the target function. The 31-bit *Config* field denotes the select lines (marked **CMx** and B**Px** in Fig. 4) of the multiplexers placed in the datapath. Two sets of multiplexers, identified as the Configuration and Bypass, respectively, are used in the datapath and allow different connection topologies between RUs. Additional configuration multiplexers are placed between the tiles to

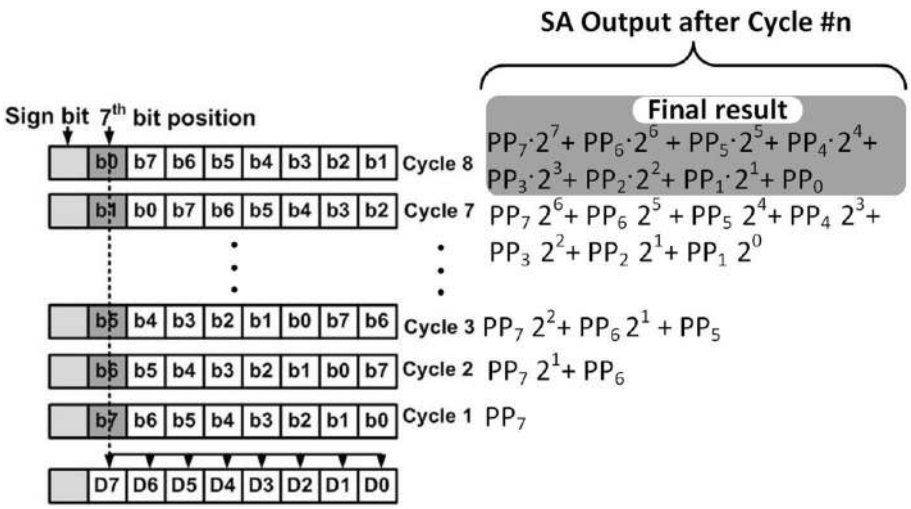

Fig. 3 MSB rotation in Coefficient Register

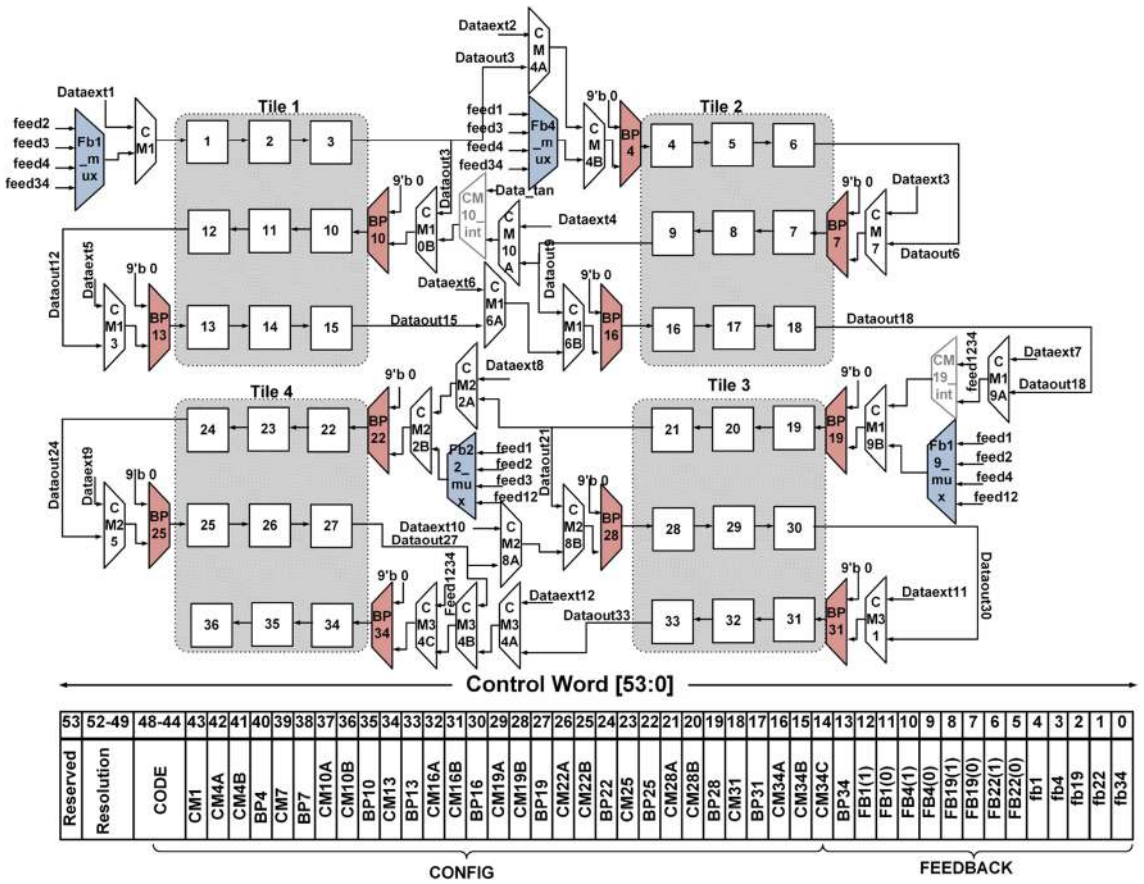

Fig. 4 The Configuration and Bypass Multiplexers in SAC Architecture

provide feedback (discussed further in Sect. 4). The bypass multiplexers are used to isolate unused RUs from the datapath and can bypass a set of three RUs at a time.

The least significant thirteen bits in the control word, labelled FBx and fbx, represent the select lines of feedback multiplexers (marked **FBx_mux** in Fig. 4) and constitute the *feedback* field of control word. Bits 0-4 of the control word are the select lines (marked **fb** in Fig. 1b) of the multiplexer inside RU that forwards registered or unregistered multiplicand. These bits are dedicated for RUs that receives feedback data, *i.e.* RU #1,4,19,22 and 34.

The leading four bits of the control word (*Resolution* field) provide the resolution of output. This is important in functions having feedback loop where the 23-b output data is limited to 9-b (width of RU data register). The resolution field indicates the relevant output bits required to be saved for further processing. The remaining bits of the control word are reserved for future modifications.

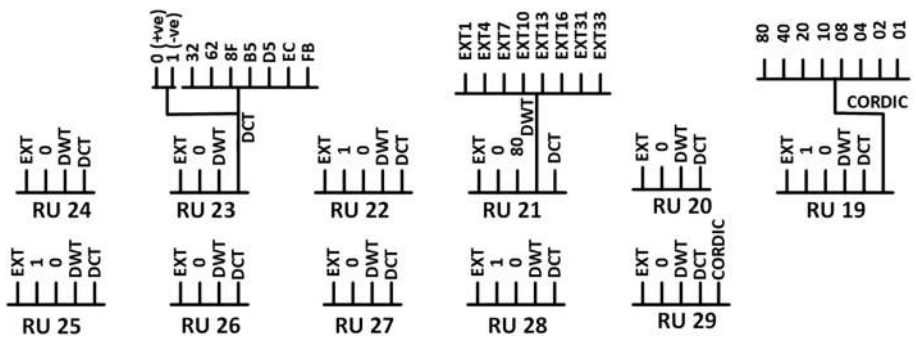

Fig. 5 The Coefficient Multiplexers in the Architecture (Analogous Representation)

3.4.1 Coefficient multiplexers

While mapping, different set of fixed coefficients are applied on RUs. For example, DCT and DWT forces their respective coefficients on RU# 19-30 as discussed in Sect. 5. The RUs where multiple coefficients are forced including the fixed coefficients are indicated in Fig. 5. For example, the DCT line is selected while mapping the DCT and is connected to DCT multiplexer that provides DCT transform matrix coefficients. The transform matrix has seven distinct coefficients that appear in either positive or negative form. Wavelet transform coefficients can be applied as external coefficients as the hardware supports multiple wavelets. A detailed discussion on coefficient multiplexers is provided in [25] and [26].

4 Architecture topologies

The configurable datapath of the SAC architecture exhibits various topologies as following:

4.1 Systolic array structure

The RUs can be arranged in a continuous chain of nine, eighteen, twenty-seven or thirty-six RUs as shown in Fig. 4. The most straight forward structure of thirty-six RUs forming a chain is formed with RUs connected one after the other. In such a case, the bypass multiplexer selects are set and configuration multiplexers pass the previous RU output. For eighteen RU, BP19 forced to '0'. A chain of nine RU is obtained by selecting tile# 1 as it consists of RU #1,2,3,10,11,12,13,14,15 by forcing '0' and '1' on CM10B and CM13, respectively. Additionally, the select line of all bypass multiplexers is set to '0' except for BP10 and BP13. Similarly, a twenty-seven RU chain is formed by combining the eighteen RU chain and tile #3 and bypassing tile #4.

4.2 Four tile structure

The four tile structure is obtained by dividing the RUs into four groups of nine RU blocks. This configuration enables parallel execution of functions as each tile forwards its partial products to their respective CU subsection that computes the results irrespective of the ongoing computation in other tile or CU subsection. The configuration multiplexers CMxB select line are forced to '0' thereby passing forward the data from the RU above it. For instance, Dataout #3, 9, 21 and 27 are fed to RU #10, 16, 28 and 34, respectively. The bypass multiplexers are set when all the four tiles are in use and a tile is isolated from the datapath by clearing the bypass multiplexer of RU leading the tile. Tiles can be connected in a chain structure that is beneficial for function chain realization where each function can be mapped on a separate tile. This can be achieved using the feedback multiplexers. Feedback multiplexers are used to direct the output (or combined output) of tile (or tiles) to input of other tiles. For instance, the four tile chain can be realized by connecting Tile #1 –> Tile #2 –> Tile # 3 –> Tile #4 by setting multiplexers **FB4_mux**, **FB19_mux** and **FB22_mux** to "00", "01" and "10", respectively. Following this topology, multiple functions are executed simultaneously while reducing the reconfiguration overhead.

Additionally, the architecture also supports mixed topologies. For instance, one of the possible combinations of systolic tile and array structures can result in datapath from Tile1 –> Tile2 –> RU #19-36. These topologies can be obtained by setting the necessary CMx and BPx bits in the control word.

5 Mapping of different functions

The configurability of the developed SAC architecture can be leveraged to realize multiple biomedical signal processing operations on it, because the majority of the functions include multiplication and addition. Mapping methodologies for these commonly used functions are presented in this section. The target functions, also optimized for the proposed hardware, are classified broadly into two categories: *Variable* and *Fixed* coefficient functions. The coefficients of variable functions depend on the characteristic parameters of the function. For example, bandwidth, gain and number of taps determine the coefficients of FIR filter. In contrast, the fixed coefficient functions have constant coefficients and they undergo fixed steps resulting in constant coefficients. Therefore, it is redundant for the user to explicitly provide constant coefficients to the hardware and is eliminated by means of simple state machines. The architecture contains state machines for all the target functions. Based on the control word, one of the state machines becomes active and controls the function execution.

5.1 Variable coefficient functions

The multiplication, squaring, addition and multiply-accumulate (MAC) functions are based on the common equation: $\sum_{i=0}^{11} a_i \cdot b_i$. A special case of MAC, where $a_i = 1$, represents the addition operation. For multiplication a_i (and/or b_i) is 0 for i>0 and for squaring, $a_i = b_i$ is done in the multiplication. The 12 RUs, marked #1, 4, 7, 10, 13, 16, 19, 21, 24, 27, 30 and 33, receive external data and computation takes eight clock cycles. It is possible to map this

operation by choosing a different set of 12 RUS anywhere in the architecture but with a higher latency.

The mathematical equation for FIR filtering, differentiation and moving average can be readily expressed as a MAC operation. However, the 2-D convolution includes the product of data and coefficient matrices as $Z(i,j) = \sum_{m=0}^{N_1-1} \sum_{n=0}^{N_2-1} X(i-m, j-n).Y(m)(n)$, where $N_1 \times N_2$ is the matrix size. The convolution consist of multiply-accumulate with spatially shifted input samples. The spatial shifts in the input matrix can be handled by careful interpretation of input rows in the memory [12]. By doing so, the 2-D convolution yields to 1-D convolution operation of length $N_1 \times N_2$. The generalized equation of FIR filtering, Differentiation, Moving Average and 2-D convolution thus becomes $\sum_{i=0}^{N-1} x[n-i] \cdot b_i$, where $N \geq N_1 \times N_2$. This output is based on the previous inputs and require memory elements. The architecture supports thirty-six taps for these set of functions. The external data ($x[n]$) is supplied on RU #1 and RUs can be connected in chain structure using the configuration and bypass multiplexers. The 2-D convolution requires input ports equal to the number of rows in the transform mask. The mask size of up to (6×6) can be mapped on the proposed hardware.

5.2 Fixed coefficient functions

Functions grouped in this category perform computations using a state machine. The state machine transforms the algorithm into a series of sequential steps. The fixed coefficients are stored in register files.

5.2.1 COrdinate Rotation DIgital Computer (CORDIC) [27]

It is derived from the general rotation transform and provides an iterative method of performing vector rotations by arbitrary angles using only shift and add operations. Generalized version of the algorithm is given by Eq. 4 [28] and includes hyperbolic and nonlinear functions, logarithm, square root computations in addition to the standard trigonometric functions.

$$\begin{aligned} x_{i+1} &= x_i - m \cdot y_i \cdot 2^{-i} \cdot d_i \\ y_{i+1} &= y_i + x_i \cdot 2^{-i} \cdot d_i \\ z_{i+1} &= z_i - \alpha_i \cdot d_i \end{aligned} \tag{4}$$

The top level hardware architecture for the CORDIC is shown in Fig. 6a. The '*X* and *Y* Computing Block' computes the *x* and *y* variables, whereas the 'Angle Accumulator Computing Block' computes the *z* variable. The

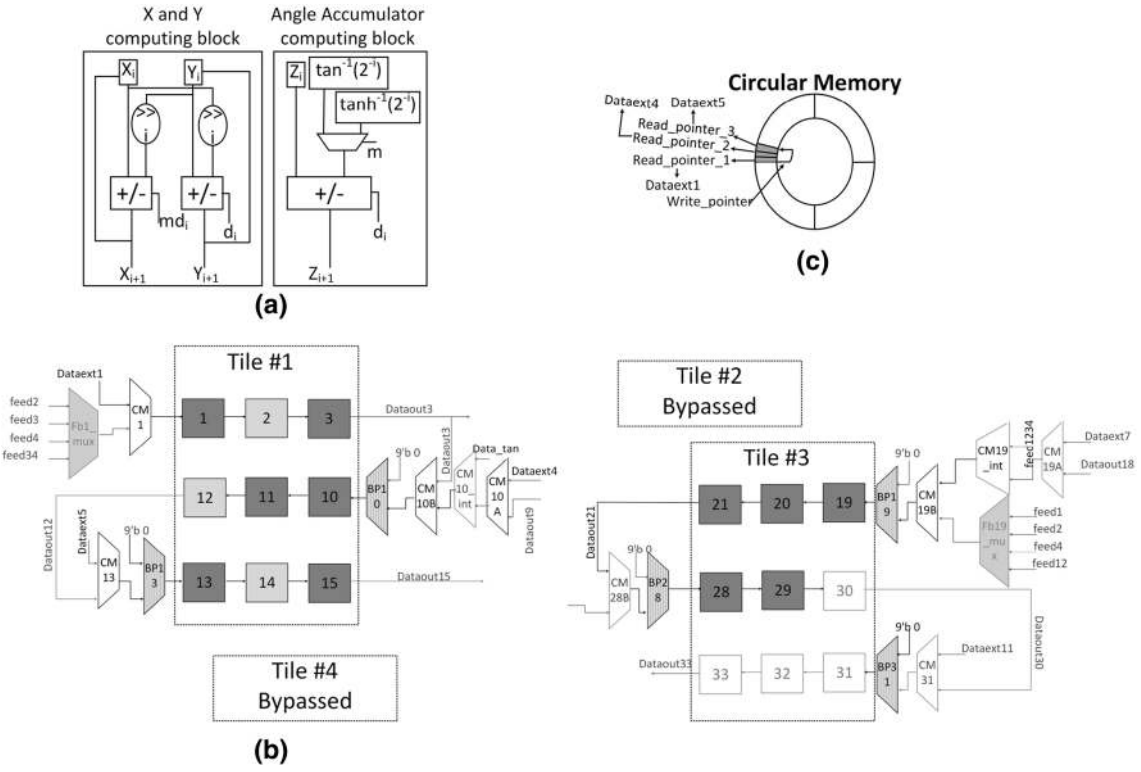

Fig. 6 CORDIC Mapping, **a** Representative CORDIC Hardware, **b** Active RUs and Datapath for CORDIC Mapping, **c** Circular Memory Structure for CORDIC

computation uses an adder or a subtractor and result from the previous iteration. The right-shift operation (>>) can be viewed as multiplication with 2^{-i}. The SAC architecture performs multiplication where the 9-b coefficient register is loaded with the content 2^{-i} in the i^{th} iteration. The data register is loaded with the x, y or z variable (or α value) and the subsequent additions are carried out in the CU. [25] provides the seed value for various functions along with m and α.

Tile #1 and tile #3 are used for CORDIC mapping (Fig. 6b). The RUs are not connected with each other in both the tiles except for RU #19-21–⟩28-30 forming a 6 RU chain. The seed values x_0, y_0, z_0 are fed into RU #1, 10 and 13 from the three read ports of circular memory (Fig. 6c). The y variable is computed first followed by z and x throughout the iterations. As y or z needs to undergo direction decision logic for next iteration, this process is overlapped with x computation so that no clock cycles are wasted. y_1 is computed using y_0 and x_0 from RU #1 and RU #13, respectively, and the result stored in RU #19. At the same time, all the RUs pushes forward its data register content to the next RU in the chain. The arctan data is forced on RU #10 through internal multiplexer (CM10_int). Variable z_1 is computed using RU #11 (z_0) and RU #10 (arctan) and z_1 is stored in RU #19, whereas y_1 shifts to RU #20. Using RU #15 (x_0) and 3 (y_0), variable x_1 is computed and stored in RU #19. At this stage, y_1, z_1 and x_1 are in RU #21, RU #20 and RU #19. y_2 is computed using RU #21 and RU #19, z_2 using RU #21 and RU #10 and x_2 using RU #21 and RU #29. It should be noted that, previous value is present in RU #21 for x, y and z. The second variable of the equation (x or y) lies in RU #19, RU #10 and RU #29. Same sequence is followed with a state machine that follows a defined pattern. The algorithmic state machine shown in Fig. 7 consists of six states and depicts the RU coefficients loaded in each state during CORDIC computation. The coefficients for RUs used in CORDIC algorithm mapping are applied through the CORDIC input of coefficient multiplexers. A CORDIC multiplexer shown in Fig. 5 is connected to the coefficient multiplexer CORDIC input of RU #19 and RU #29. The multiplexer has 2^i terms in binary as inputs and its select line is controlled by the state machine and takes 24 cycles for each iteration. It stores results internally in RUs and does not use additional memory.

5.2.2 The 2-D Discrete Wavelet Transform

2-D DWT translates to two consecutive matrix multiplication between the input and DWT mask coefficients. The results of matrix multiplication can be directly realized as multiply-accumulate operation and thus can be supported by the SAC architecture. The DWT decomposes a signal into four frequency sub-bands, namely LL (approximation (cA) matrix), LH (horizontal (cH) matrix), HL (vertical (cV) matrix) and HH (diagonal (cD) matrix). A generic mapping methodology is discussed that supports twenty-eight wavelets with filter size <8 [26]. The DWT algorithm first computes low pass filtering followed by down sampling by a factor of 2. The resultant vector is convolved with high

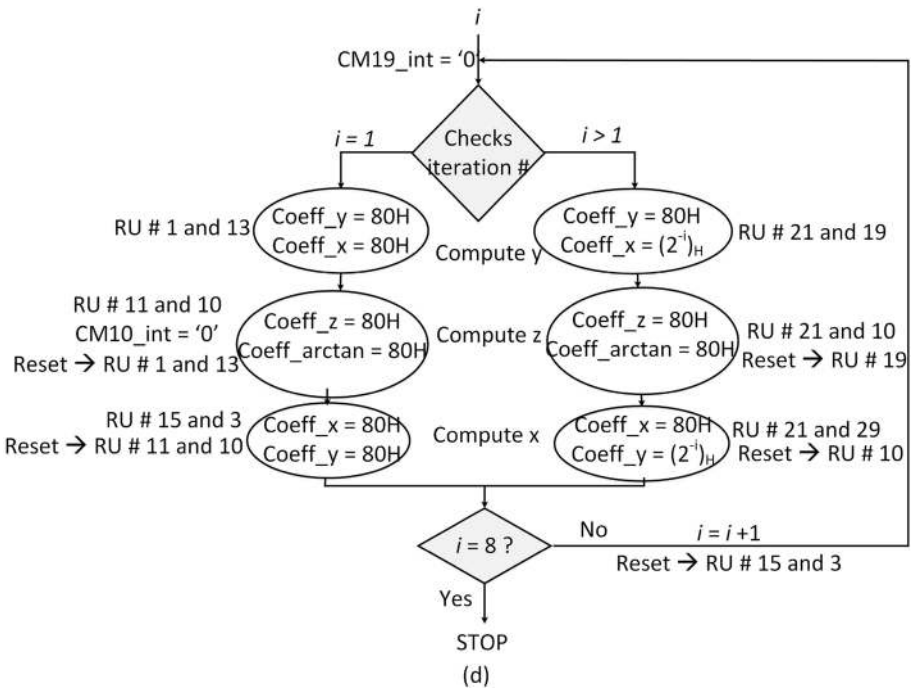

Fig. 7 Flow Chart of CORDIC State Machine

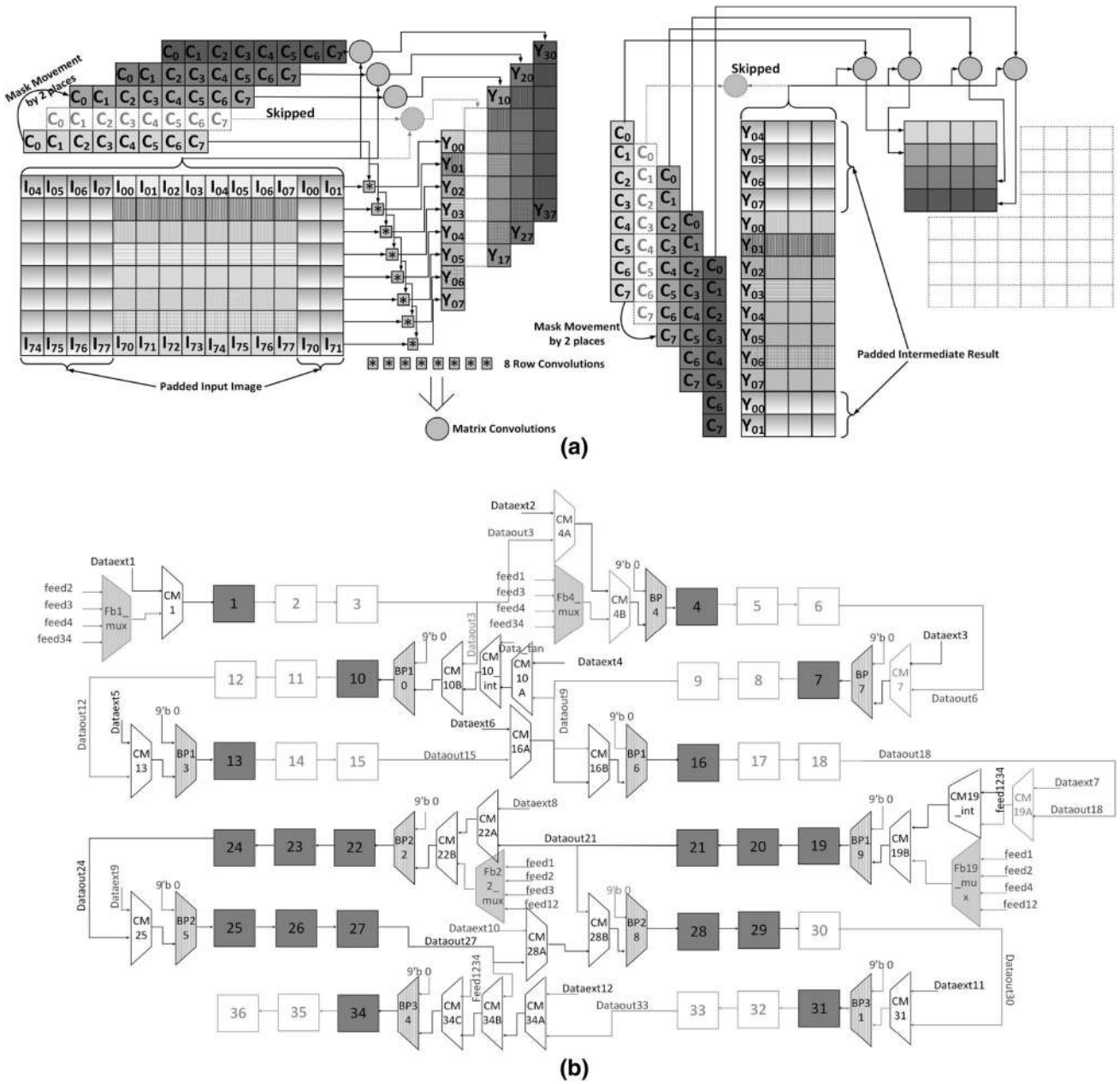

Fig. 8 DWT Mapping **a** Discrete Wavelet Transform Algorithm, **b** 2-D DWT Mapping on the SAC Architecture

pass filter and the LL DWT matrix is retained (with 75% compression) after another down sampling by a factor of 2. Additionally, periodic padding [29] is done on either side of the input samples and intermediate results to ensure spatial continuity on boundaries. The DWT computation steps are shown in Fig. 8a wherein the row wise convolution is carried out on a 14×8 padded image and an intermediate matrix, Y is generated. Columns 0-3 of padded image are same as columns 5-8 of the actual image. Thus, the convolution filter mask is modified by swapping the first four terms with the last four terms. This eliminates the use of excess memory elements required to store the padding. Furthermore, the mask advances by two steps as opposed to one step in the conventional convolution performing down sampling along with the intermediate matrix generation. The second set of convolution is performed on transposed padded Y matrix.

The proposed architecture supports 8×8 block 2-D DWT operation with the mapping scheme are indicated in Fig. 8b. The five states of the DWT state machine are shown in Fig. 9a. The intermediate matrix (Y) rows are computed in the first state and the other four states

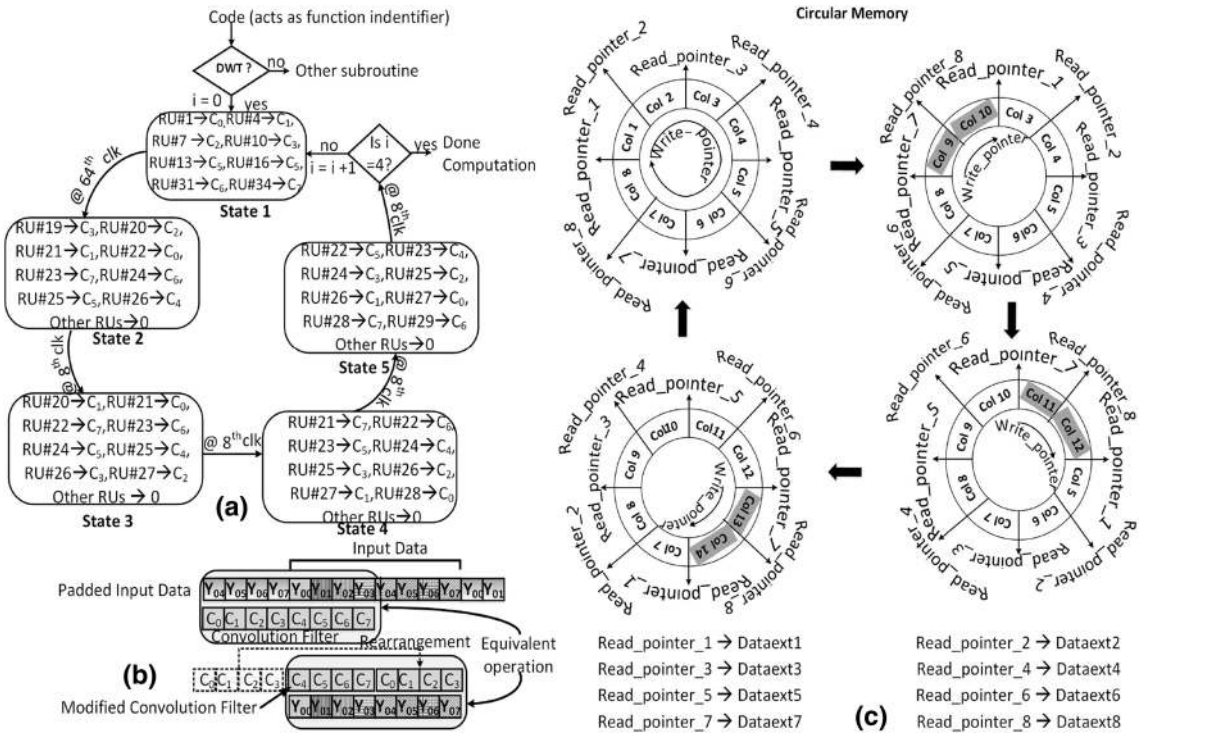

Fig. 9 DWT Mapping **a** Flow Chart of 2-D DWT State Machine, **b** DWT Mask Manipulation, **c** Data Stored in Interleaved Manner in the Circular Memory with the Write and Read Pointers. The updated columns are indicated by grey filled text

compute the first row of DWT matrix. Convolution filter coefficients provided by the state machine are input to RU #1, RU #4, RU #7, RU #10, RU #13, RU #16, RU #31 and RU #34 and are multiplied with the first row of image matrix to generate the first element of Y matrix. Similarly, the rest of the seven elements of first row of Y matrix are generated in 64 cycles. RU #19-RU #26 holds the first row which requires padding on either sides. This is done by means of convolution mask rearrangement. The convolution filter is rearranged by moving the first four coefficients towards the end. By doing this, the padded elements which are at the bottom of unpadded image gets multiplied with the respective coefficients yielding the same result as padded image with the unmodified filter mask (Fig. 9b). The mask manipulation enables computing cA elements without external padding on the intermediate matrix which further permits storing the intermediate matrix. The circular memory write pointer is used to write 8×8 (= 64) input data which is stored column-wise in different sections of the memory in an interleaved manner (Fig. 9c) for simplifying the reading operation. It also ensures moving the convolution mask by 2 steps, a scheme adopted in the mapping to eliminate down sampling later. Consequently, multiple read ports ensure reading all columns simultaneously resulting in computation of Y matrix to be computed in eight clock cycles.

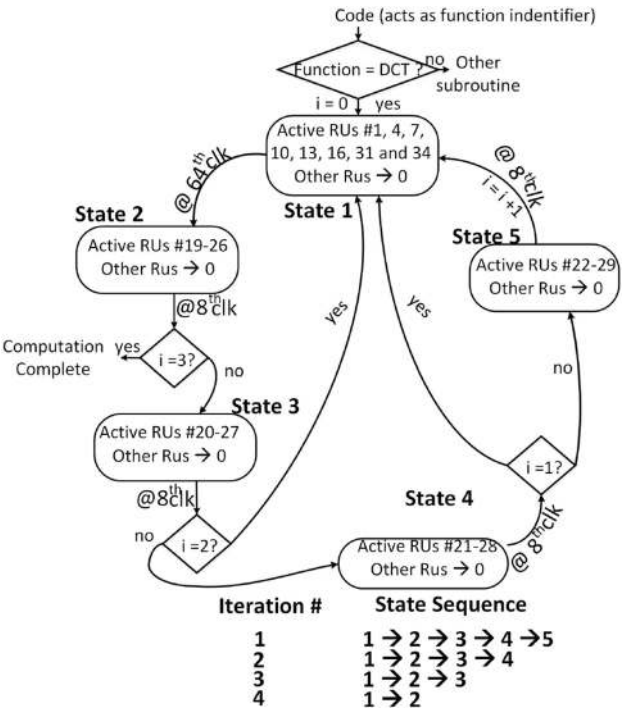

Fig. 10 Flow Chart of 2-D DCT State Machine

5.2.3 The 2-D discrete cosine transform (DCT)

Matrix computation (of an image) requires two matrix multiplication operations : DCT = DCT_{mat} * Image * $(DCT_{mat})^T$, where transform matrix (DCT_{mat}) becomes constant for a fixed mask size. Matrix multiplication can be realized on the proposed architecture by loading row and column elements of the two matrices in place of coefficients and data registers, respectively. The architecture supports 8×8 block wise DCT and provides a compression of ≈ 84% (10 of 64 output samples are retained). The DCT mapping datapath is similar to the DWT mapping methodology shown in Fig. 8b. The first matrix multiplication or the intermediate matrix Int_{mat} = DCT_{mat} * Image, row wise image data and first row of DCTmat are forced on RU #1, RU #4, RU #7, RU #10, RU #13, RU #16, RU #31 and RU #34 as data and coefficient, respectively. The results form the first row of Int_{mat} that is periodically stored from RU #19-RU #26 and has to be further multiplied with $DCT_{mat}{}^T$.

The row coefficients of the transform matrix are forced on coefficient registers of RU #19-RU #26 and the resultant DCT(0,0) is obtained. On forcing subsequent DCT_{mat} rows in the coefficient of RU #19-RU #30 chain, the remaining elements of the first row of DCT are obtained. It should be noted that, Int_{mat} stored in RU #19-RU #26 shifts down the RU chain every 8 clock cycles. This is incorporated by the state machine (Fig. 10) that consists of 5 states traversed four times, fully or partially to obtain Int_{mat} and DCT in successive computations. The state sequence differs with the iteration#, because the number of elements computed in the DCT matrix decreases by one with every row resulting in an upper triangular matrix. The Int_{mat} matrix is computed in state #1 and DCT matrix is computed in states #2-5. The image data is stored column wise in an interleaved manner in circular memory similar to data storage in DWT mapping discussed earlier.

Table 3 Control Word for Target Functions and Topologies

Function	Code	Config
Multiplication	01	00000000
Square	02	00000000
Addition/MAC	03/07(3-tap)	00D00000
	04/08(6-tap)	0AD60000
	05/09(9-tap)	0AD640D
	06/0A(12-tap)	0AD66AB7
FIR,$\frac{d}{dx}$, MA	0B(9-tap)	00700000
	0C(18-tap)	2FFE0000
	0D(27-tap)	2FFF4070
	0E(36-tap)	2FFF6FFF
DWT	0F	0AB66FD7, fb19='1'
2-D Convolution	10	00D00000
CORDIC	11	00D04040, fb19='1'
DCT	12	0AB66FD7, fb19='1'
Topology	Config	Feedback
Four Tile	0E724E73	–
Systolic Tiles		
#1–>#2	1E720000	0008
#1–>#2–>#3	1E72C070	008C
#1–>#2–>#3–>#4	1E72DE73	00CE
RU chain–>#3–>#4	2FFEDE73	01C6

6 Results and discussion

To demonstrate the feasibility of the proposed architecture, the design was targeted onto a Virtex-II FPGA board (XUPV2P). The FPGA development board speeds up the verification process by providing suitable interfaces in VGA and RS-232 ports among many other peripheral interfaces. The proposed architecture uses the sign-magnitude convention to represent 9-b signed numbers. The *code*, *config* fields and topologies for targeted functions are presented in Table 3, where the *code* acts as target function identifier and controls the data read write operations of circular memory. The feedback gives rise to erroneous outputs as fixed-point binary numbers are used. This is addressed by making the resolution configurable that detects the correct output slice fed back to the RUs.

6.1 Hardware configuration

The architecture is configured by loading the control word, external coefficients and data in their respective memories. These values are provided to the architecture through a 9-b input bus in a time multiplexed manner. The 54-b control word is applied first serially in 9-b slices. The external coefficients are loaded next and comprise of 36 9-b binary numbers. Following this, input bus carries the data designated for the 9-b wide 64 position deep data memory. The input bus is controlled by mutually exclusive control signals that are used to demarcate the control word, coefficients and data.

The system level block diagram of the proposed architecture is shown in Fig. 11. The architecture has multiple memory modules for configuration, control word, coefficient, data and output. The configuration memory is initialized using a coefficient file (.coe) that can be generated using MATLAB for large data sets. Data memory has an adaptable structure to support varied requirements of the target functions. A UART serial port (bit rate set at 115200 bps) is interfaced with the hardware to transfer the output from FPGA to the PC for post processing.

Table 4 Transfer function of PTA blocks [16] with their Mapping on SAC Architecture

Block Name	Transfer function	Code	Control word config	Datapath (RUs)
LPF	$(1\text{-}2z^{-6}+z^{-12})/(1\text{-}2z^{-1}+z^{-2})$	0C	2FF000010001	#1– #13, #34 – #35
HPF	$(\text{-}1+32z^{-16}+z^{-32})/(1+z^{-1})$	0E	2FFF6FF10001	#1– #33, #34
Differentiator	$(z^{-2}\text{-}2z^{-1}+2z^{1}+z^{2})/8$	0B	280000000000	#1– #5
Squaring	$x[n]^2$	02	000000000000	#1
Moving Average	$\frac{1}{32}\sum_{i=0}^{31} x[n-i]$	0E	2FFF6FF10000	#1 – #32

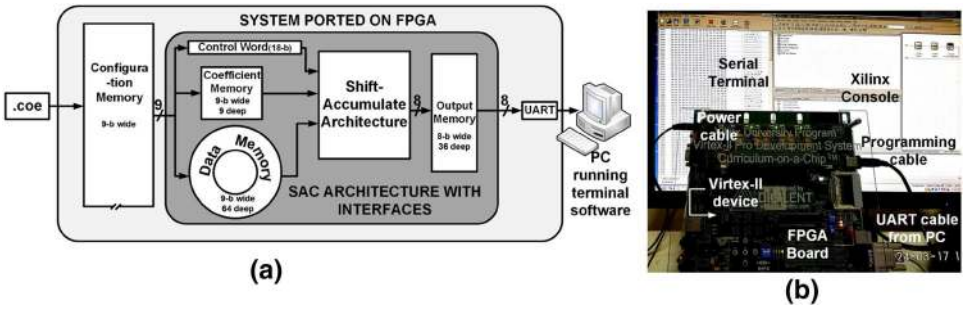

Fig. 11 **a** System Level Interaction of SAC Architecture with Peripherals, **b** Actual Hardware Setup

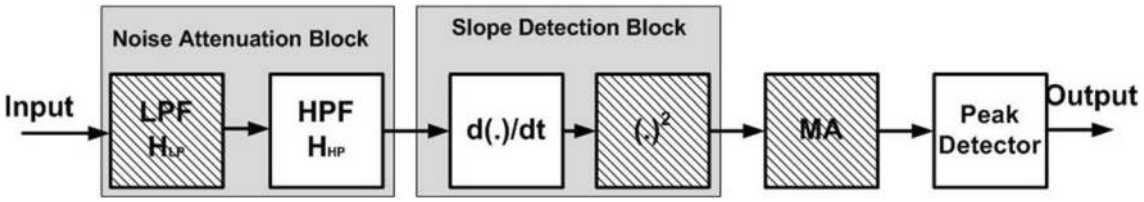

Fig. 12 PTA Signal Chain

6.2 Pan–Tompkins algorithm signal chain

To verify the usefulness and functionality of the proposed hardware, we have targeted a popular algorithm known as the Pan–Tompkins algorithm (PTA). This is a widely used algorithm for QRS detection in Electrocardiogram (ECG) signals that detect QRS complexes with accuracy up to 99.3% [16]. The ECG signal from the MIT-BIH arrhythmia database is given as input to SAC architecture after converting it to .coe format. The algorithm is based on analysing the amplitude, slope and width of QRS complexes and is modelled with steps shown in Fig. 12. The transfer function of the steps is shown in Table 4.

Careful observation of the transfer functions of QRS signal chain indicates that LPF and HPF (collectively BPF), differentiation, squaring and MA functions can be realized by a series of shift-accumulate operations resulting in multiplier-less implementation of these functions. The control word and datapath for PTA signal chain is tabulated in Table 4. The proposed architecture is first configured as a LPF and the result is computed in 8 clock cycles. Following this, the architecture is configured to perform HPF. It should be noted that the HPF operates upon the LPF output. Therefore, the LPF output is loaded into the circular memory which further acts as the input data to the proposed architecture. The HPF output computation takes 8 clock cycles. The architecture is further configured to emulate differentiation, squaring and MA functions. The control word and the coefficients of these functions are stored in the configuration memory that is accessed using a state machine after each operation.

The LPF block has a window of 13 inputs with feedback of 2 output samples. RU #1 accepts the input data from *Dataext1* port and a chain of 13 RUs (RU #1 – #13) is formed. The configuration multiplexers are configured to forward the data from previous RU and bypass multiplexers are set. The output samples are loaded into RU #34 and #35 through the *feed1234* dataline. Control word bit 0 (*fb34*) is set as RU #34 carries feedback data. The remaining RUs are disconnected from the datapath using bypass multiplexers. The computed hardware result for each block is shown in Fig. 13. The peaks obtained in the moving average output represent the QRS complex that can be detected further using a peak detector circuit. It can be seen that minor peaks are flattened in hardware results due to limited resolution in feedback. The QRS peaks

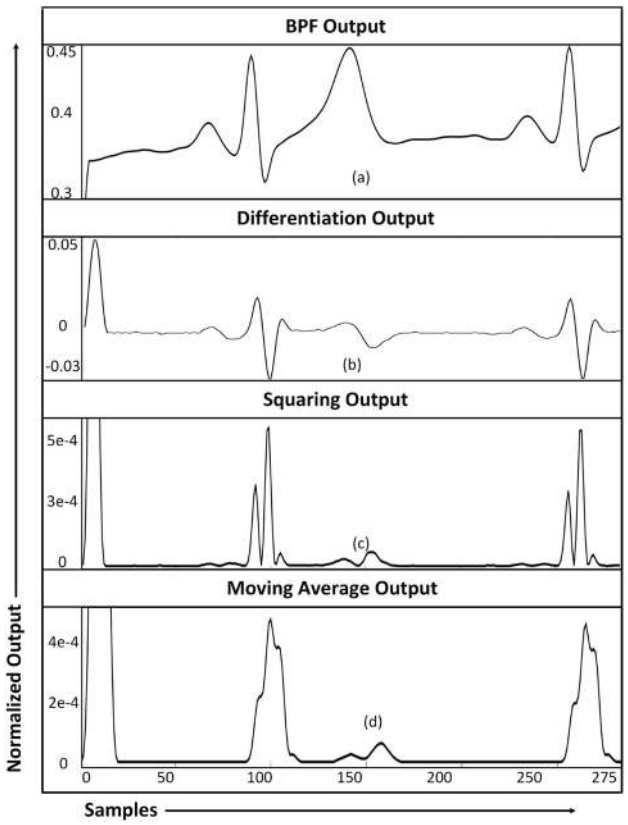

Fig. 13 Hardware Computed PTA Results **a** BPF. **b** Differentiation. **c** Squaring, **d** Moving average

Table 5 Range and Resolution of CORDIC on Developed Architecture

Functions	H/W Input Range	HDL resolution	
		Output	Input
sin, cos	−90° to +90°	2^{-7}	2^{-1}
sinh , cosh, *exp*	−1.118 to +1.118	2^{-7}	2^{-6}
ln	0.1068 to 9.360	2^{-6}	2^{-4}

detected on the hardware processed data coincides with the analytically detected peaks. The architecture takes 485 clock cycles/sample (= 10.34 μs or throughput 96.7e3 samples/second at 46.9 M*Hz* operating frequency), including configuration, data loading and processing, while emulating the PTA signal chain. Biomedical signal processing applications pose moderate processing requirements on the computing platform primarily due to low data rates (up to 320 Kb/s) [30] of physiological signals. Additionally, according to a study in [31], the operating frequency of the processing platform are limited to 10's of k*Hz* and (8-12)-bit processing for biomedical devices. This leaves sufficient clock cycles to tolerate the latency of the serial SAC architecture of the ECG application mentioned above that has been demonstrated to take 10.34 μs to process a single data sample. Also, SAC architecture supports 8-b signed arithmetic that has been adopted in various matured applications in the domain [31]. However, there is scope for incorporating the applications requiring increased precision by increasing the number of bits in the data/coefficient registers and scaling up the adder units accordingly. It should be noted that, we have not considered Electromagnetic Interference of ECG signals in our setup. The work is primarily focused on developing a configurable lightweight processing platform for biomedical signal processing algorithms. The EMI and other artefacts are assumed to be taken care of by the acquisition system which is beyond the scope of this work. We used the ECG signal from the MIT-BIH arrhythmia database is given as input to SAC architecture. The ECG signal is collected from the patients in normal clinical setting and are not filtered. The type of noises in the database is not explicitly mentioned.

6.3 CORDIC, DCT and DWT results

Table 5 notes the range and resolution supported by the hardware for CORDIC. The results obtained from the hardware agrees well with the ideal (MATLAB) results for functions such as sin(cos); sinh(cosh); e^x and ln(x) and are shown over their respective range shown in Fig. 14a–d. The architecture computes cosine with accuracy ranging from 83 to 98% and the resolution is 2^{-1}(= 1 bit) when the decimal point is between 1^{st} and 0^{th} bit position. Thus, the smallest measurable degree is 0.5° while the remaining 7 bits denotes range of the data.

The test setup for the DCT is shown in Fig. 15a. It uses a 128×128 image as input. The 8×8 sub-images undergo the DCT operation on the proposed architecture and results are processed in MATLAB which reconstructs the image by performing offline block wise IDCT. The reconstructed image obtained from the hardware and MATLAB are shown in Fig. 15b and c, respectively. The L2 norm is reported as 15.77 and 15.75 for hardware and MATLAB reconstructed images, respectively, with respect to the actual image.

To demonstrate the feasibility of the proposed approach, the 2-D DWT with two standard test images is taken. A 16×16 checker board image with Haar wavelet and a 128×128 Lena image with bior2.2 wavelet is computed on the hardware. The mean square error between the actual and hardware reconstructed image for the Lena intensity image is 2.2*m*. The actual and reconstructed Lena images both from MATLAB simulation and hardware implementation are shown in Fig. 16. The image metrics are reported in Table 6. The MSE for reconstructed images

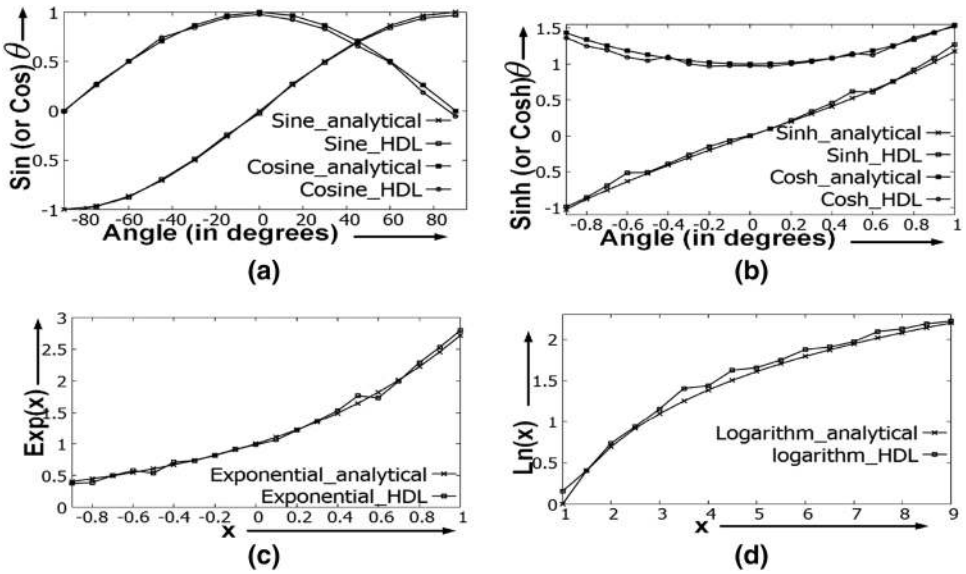

Fig. 14 The Hardware and MATLAB Results **a** Sine and Cosine **b** Hyperbolic Sine and Cosine **c** Exponential, **d** Ln Functions

Table 6 Comparison of the actual and reconstructed lena images

Image	Reconstructed image	PSNR(dB)	Mean square error	L2-Norm
Actual	MATLAB	74.8819	0.0020	0.9904
Actual	Hardware	74.6535	0.0022	0.9514
Reconstructed Images				
MATLAB	Hardware	87.7629	1.0883e-04	0.9606

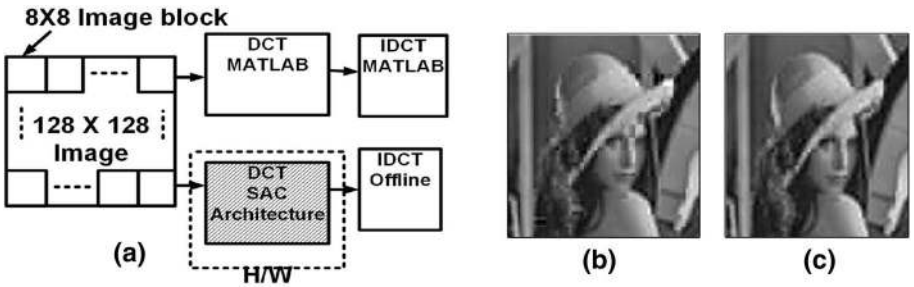

Fig. 15 DCT **a** Hardware setup block diagram, **b** Hardware and **c** MATLAB Results

Fig. 16 DWT Results: **a** Actual, **b** MATLAB, **c** Hardware Computed

Table 7 The phase wise cycle counts for target functions

Functions	Configuration	Cycle Counts# Coefficient load	Data Load	Compute	Throughput (samples processed/second)
Multiplication, Square	6	1	1	8	2.9e6
Addition, MAC (3-12 taps)	6	3-12	3-12	8	2.3e6-1.2e6
FIR, MA, Differentiation (9-36 taps)	6	9-36	64	**8**	5.4e5-4.1e5
2-D DWT	6	8	64	**496**	81.7e3
2-D Convolution	6	36	64	**8**	4.1e5
CORDIC	6	1	3	**24**	1.3e6
2-D DCT	6	1	64	**336**	1.1e5

cycles marked in bold are per sample (FIR), per iteration (CORDIC), per 8×8 data block (DWT, DCT)

is within 10*m* range with 87.7629 PSNR indicating faithful reconstruction of the hardware image.

6.4 Clock profiling

The clock profile can be broadly divided into configuration and computation phases. The phase wise clock profile of the architecture along with the throughput of target functions is provided in Table 7. Configuration phase includes control word, coefficient and data loading through the 9-bit input bus. The 54-bit control word is loaded serially and takes 6 clock cycles.

Coefficient Load Latency The coefficient load latency depends on the nature/amount of processing involved in computing the function. The FIR computation, addition and MAC require coefficients equal to the number of filter taps/terms involved in computation. Therefore, the coefficient load latency is indicated as 9-36 and 3-12 clock cycles, respectively. In case of DWT, eight filter coefficients are loaded in 8 clock cycles. The CORDIC and DCT coefficients (8×8 mask size) are fixed and stored within the architecture, thus are applied through their respective state machines in 1 clock cycle. The 2-D convolution latency is indicated as 36 clock cycles for the 6×6 mask, *i.e* 36 coefficients.

Data Load Latency The data load latency depends on the input data requirement of the target functions. The data is loaded into the architecture from the circular memory and take 1 clock cycle for each load operation. The addition and MAC require data equal to the number of terms in the expression and thus takes 3-12 clock cycles. The entire circular memory is loaded in case of sliding window functions (FIR, MA, differentiation, 2-D Convolution) resulting in a latency of 64 clock cycles. For CORDIC computation, the three variables require seed values to initiate the computation and thus the data load latency is 3 clock cycles. DCT and DWT support operation on 8×8 sub-blocks and thus takes 64 clock cycles while loading 64 data.

Compute Latency Target functions include functions readily expressed as multiply-accumulate (FIR, Convolution, Multiply, Addition, Multiply-accumulate, Moving Average) and functions that are interpreted as multiply-accumulate after data/algorithm manipulations by means of state machine (CORDIC, DCT, DWT). The compute latency for the former set of functions is 8 clock cycles. In the later set of functions, CORDIC takes 24 cycles, DCT takes 336 cycles and DWT takes 476 cycles for computations following the state machines shown in Figs. 6d, 10 and 9a. The CORDIC computation

Table 8 Comparison with the State of the Art

	[8]	[1]	[32]	This work
Equivalent Gate Count	81.3k	36k	195k	24,280
VDD[V], *f*[MHz]	1,10	0.34,0.6	0.4,1	1.8,46.9
Target functions	32-tap FIR, 65-pt median, 512-pt FFT, CORDIC	LPF,HPF, Derivative-square, moving average, peak detection	CWT-based QRS detection	FIR, Moving Average, CORDIC, DCT, DWT, Multiplication, Addition, MAC, Square
Latency (in cycles)	***32-88***	***N.A.***	***N.A.***	***16-574***
On-Chip Memory	64kb	-	2Mb	954b
Platform	Hardware accelerators	Custom μP	CoolFlux BSP	Custom Re-configurable HW

is explained here as an example. The CORDIC algorithm requires computation of three variables in every iteration. The equation for each of the variables includes multiple-accumulate operation, thus takes 8 clock cycles, and are computed serially as the past variables are used in the computation of present variables. This amounts to the total latency of 24 clock cycles.

6.5 Gate profiling

The total number of gates in the architecture is found by estimating gates in its individual blocks, *i.e.* RUs and CU. Each RU contains 9 XOR, 8 AND, 27 flip-flops and 1 2:1 multiplexer totalling 242 gates. There are 208 equivalent 1-bit adders in CU along with 42 flip-flops amounting to 4376 gates. The configurable datapath multiplexers have 396 equivalent 2:1 multiplexer, whereas coefficient multiplexers are the largest contributor with 9608 gates owing to large DCT, DWT and CORDIC multiplexers. This brings the proposed architecture gate count to 24280 gates. The interfacing memory has 954 flip-flops adding another 6678 gates to the system.

6.6 Comparison with similar works

The architecture developed in this work acts as a multiple hardware accelerator because of its configurable datapath and adopts effective mapping methodologies translating to gate and clock savings. The comparison of the proposed architecture with the state of the art is provided in Table 8. The work described in [8] advocates the use of accelerators dedicated for commonly used signal processing operations like FIR, CORDIC, moving average integration, *etc*. The biomedical signal processor in [8] makes use of the aforementioned functions and the gate count reported for them is 11k, 9.3k and 37k, respectively, with operating frequency of 1MHz for individual execution of these functions. [1] implements ECG signal processing algorithm on a custom microprocessor and reports 36k gates at an operating frequency of 600kHz.

A similar system operating at 1M*Hz* in [32] reports a gate count of 195k gates The proposed work reports a gate count of 24280 gates providing ≈1.5× to 8× gate (and area) savings with respect to [1] and [32], respectively.

The latency comparison indicates that accelerators in [8] take 32–88 cycles for computing various target functions, whereas the SAC architecture takes 16-574 cycles (0.34-12 ns 46.9M *Hz*). In both cases, the latency includes the cycles taken in supplying data, transferring output and configuring the architecture. The increased latency in this work is primarily due to the serial nature of the proposed architecture; however, cycle optimizing techniques focused on restricting the data movements are adopted while developing the mapping methodologies. In general, biomedical signal processing applications expect moderate processing rates from the processing platforms due to the nature of biomedical signals (data rates up to 320Kb/s [30]) as well as the data manipulation techniques. This ensures that the upper bound of operating frequency of SAC architecture was sufficient to support a variety of biomedical algorithm processing.

7 Conclusion

A shift-accumulate-based architecture is developed with configurable datapath that supports dominant DSP functions for biomedical signal processing. The architecture exhibits different topologies for efficient realization of functions. Efficient mapping methodologies are developed for FIR, CORDIC, DWT, DCT, *etc.*, by exploiting algorithm regularities that ensure cycle efficient calculations by eliminating redundant computations and limiting memory requirements. Additionally, SAC architecture contains a configurable datapath that can be leveraged to support algorithm advancements. Additionally, the architecture supports modularity, *i.e.* multiple SAC architecture units can be connected to exercise various topologies and functionalities.

The architecture is targeted on a state-of-the-art FPGA and demonstrates multiple functions like PTA signal chain, CORDIC, DWT, DCT, *etc*. The maximum operating frequency supported by the architecture is 46.9M*Hz*. The architecture computes cosine with accuracy ranging from 83 to 98% and a mean square error of ≈0.3 is reported for Lena image as compared to the uncompressed image. The architecture takes 16-574 cycles for target functions. In addition, the gate count of the architecture is estimated as 24280 gates and offers 8× area advantage making it suitable to be deployed in biomedical applications. The FFT mapping scheme on the architecture is planned as future work. Additional benefits of further improvements in power consumption utilizing dedicated power domains have been proposed in [32, 33] yielding energy advantages. The applicability of multiple power domain to the proposed architecture is left as future work. Furthermore, the proposed hardware can benefit from energy efficiency ensured by approximate computing and is also left as future work.

Acknowledgements The authors would like to thank Palas Parmar and Mansi Singh for their assistance in DCT and DWT analysis algorithm.

Declarations

Conflict of interest The authors declare that they have no conflict of interest

References

1. Abdallah RA, Shanbhag NR (2013) An energy- efficient ECG processor in 45-*nm* CMOS using statistical error compensation. IEEE J Solid-State Circuits 48(11):2882–2893. https://doi.org/10.1109/JSSC.2016.2542116
2. Jain SK, Bhaumik B (2017) An energy efficient ECG Signal processor detecting cardiovascular diseases on smartphone. IEEE Trans Biomed Circuits Syst 11(2):314–323. https://doi.org/10.1109/TBCAS.2016.2592382
3. Verma N, Shoeb A, Bohorquez J, Dawson J, Guttag J, Chandrakasan AP (2010) A micro-power EEG acquisition SoC with integrated feature extraction processor for a chronic seizure detection system. IEEE J Solid-State Circuits 45(4):804–816. https://doi.org/10.1109/JSSC.2010.2042245
4. Li P, Zhang X, Liu M, Hu X, Pang B, Yao Z, Jiang H, Chen H (2017) A 410-*nW* efficient QRS processor for mobile ECG Monitoring in 0.18μm CMOS. IEEE Trans Biomed Circuits Syst 11(6):1356-1365. https://doi.org/10.1109/TBCAS.2017.2731797
5. Zhang Y, Tian Y, Wang Z, Ma Y, Ma Y (2013) An ECG intelligent monitoring system with MSP430 Microcontroller. In: 2013 8th international workshop on systems, signal processing and their applications (WoSSPA). PP. 214-219. https://doi.org/10.1109/WoSSPA.2013.6602364
6. Kim H, Yazicioglu RF, Torfs T, Merken P, Van Hoof C, Yoo HJ (2009) An integrated circuit for wireless ambulatory arrhythmia monitoring systems. In: 2009 annual international conference of the IEEE Engineering in Medicine and Biology Society. pp 5409–5412. https://doi.org/10.1109/IEMBS.2009.5332815
7. Sridhara SR, DiRenzo M, Lingam S, Lee SJ, Blazquez R, Maxey J, Ghanem S, Lee YH, Abdallah R, Singh P, Goel M (2011) Microwatt embedded processor platform for medical system-on-chip applications. IEEE J Solid-State Circuits 46(4):721–730. https://doi.org/10.1109/JSSC.2011.2108910
8. Kwong J, Chandrakasan AP (2011) An energy-efficient biomedical signal processing platform. IEEE J Solid-State Circuits 46(7):1742–1753. https://doi.org/10.1109/JSSC.2011.2144450
9. Baquero JS, Cabrera FL, de Sousa FR (2016) A miniaturized low-power radio frequency identification tag integrated in CMOS for biomedical applications. In: 2016 1st international symposium on instrumentation systems, circuits and transducers (INSCIT), pp 10–13. https://doi.org/10.1109/INSCIT.2016.7598209
10. Wu J, Li F, Chen Z, Pu Y, Zhan M (2019) A neural network-based ECG classification processor with exploitation of heartbeat similarity. IEEE Access 7:172774–172782. https://doi.org/10.1109/ACCESS.2019.2956179
11. Wei Y, Zhou J, Wang Y, Liu Y, Liu Q, Luo J, Wang C, Ren F, Huang L (2020) A review of algorithm & hardware design for AI-based biomedical applications. IEEE Trans Biomed Circuits Syst 14(2):145–163. https://doi.org/10.1109/TBCAS.2020.2974154
12. Sunwoo MH, Oh SK (2004) A Multiplierless 2-DConvolver chip for real-time image processing. J VLSI Signal Process Syst Signal Image Video Technol 38(1):63–71. https://doi.org/10.1023/B:VLSI.0000028534.35761.a8
13. Mishra B, Al-Hashimi BM (2008) Subthreshold FIR filter architecture for ultra low power applications. In: International workshop on power and timing modeling, optimization and simulation, pp 1–10. https://doi.org/10.1007/978-3-540-95948-9_1
14. Rusch TL, Sankar R, Scharf JE (1996) Signal processing methods for pulse oximetry. Comput Biol Med 26(2):143–159. https://doi.org/10.1016/0010-4825(95)00049-6
15. Emmanuel BS (2012) A review of signal processing techniques for heart sound analysis in clinical diagnosis. J Med Eng Technol 36(6):303–307. https://doi.org/10.3109/03091902.2012.684831
16. Pan J, Tompkins WJ (1985) A real-time QRS detection algorithm. IEEE Trans Biomed Eng 3:230–236. https://doi.org/10.1109/TBME.1985.325532
17. Zong W, Moody GB, Jiang D (2003) A robust open-source algorithm to detect onset and duration of QRS complexes. Comput Cardiol 2003:737–740. https://doi.org/10.1109/CIC.2003.1291261
18. Kim J, Shin H (2016) Simple and robust realtime QRS detection algorithm based on spatiotemporal characteristic of the QRS complex. PLoS ONE 11(3):1–13. https://doi.org/10.1371/journal.pone.0150144
19. Arefin MR, Tavakolian K, Fazel-Rezai R (2015) QRS complex detection in ECG signal for Wearable Devices. In 2015 37th annual international conference of the ieee engineering in medicine and biology society (EMBC), pp 5940–5943. https://doi.org/10.1109/EMBC.2015.7319744
20. Martínez JP, Almeida R, Olmos S, Rocha AP, Laguna P (2004) A wavelet-based ECG delineator: evaluation on standard databases. IEEE Trans Biomed Eng 51(4):570–581. https://doi.org/10.1109/TBME.2003.821031
21. Shoeb A, Edwards H, Connolly J, Bourgeois B, Treves ST, Guttag J (2004) patient-specific seizure onset detection. Epilepsy Behav 5(4):483–498. https://doi.org/10.1016/j.yebeh.2004.05.005
22. Chen D, Wan S, Xiang J, Bao FS (2017) A high-performance seizure detection algorithm based on discrete wavelet transform (DWT) and EEG. PLoS ONE 12(3):1–21. https://doi.org/10.1371/journal.pone.0173138
23. Subasi A (2005) Automatic recognition of alertness level from EEG by using neural network and wavelet coefficients. Expert Syst Appl 28(4):701–711. https://doi.org/10.1016/j.eswa.2004.12.027
24. Chan V, Underwood S (2005) A single-chip pulse oximeter design using the MSP430. https://www.ti.com/lit/an/slaa274b/slaa274b.pdf
25. Jain N, Mishra B (2015) CORDIC on a configurable serial architecture for biomedical signal processing applications. In 2015 19th International symposium on vlsi design and test, pp 1–6. https://doi.org/10.1109/ISVDAT.2015.7208050
26. Jain N, Singh M, Mishra B (2018) Image Compression using 2 D-discrete wavelet transform on a light-weight reconfigurable hardware. In 2018 31st international conference on VLSI design and 2018 17th international conference on embedded systems (VLSID), pp 61–66. https://doi.org/10.1109/VLSID.2018.38

27. Volder JE (1959) The CORDIC trigonometric computing technique. IRE Trans Electron Comput 8:330–334. https://doi.org/10.1109/TEC.1959.5222693
28. Walther JS (1971) A unified algorithm for elementary functions. In: Proceedings of 1971 spring joint computer conference, pp 379–385. https://doi.org/10.1145/1478786.1478840
29. Skodras A, Christopoulos C, Ebrahimi T (2001) The JPEG 2000 still image compression standard. IEEE Signal Process Mag 18(5):36–58. https://doi.org/10.1109/79.952804
30. Biopotential Nagel JH, Amplifiers (2000) In: Bronzino JD (ed) The biomedical engineering handbook: medical devices and systems. CRC Press LLC, pp 1–14
31. Chandrakasan AP, Verma N, Daly DC (2008) Ultra low-power electronics for biomedical applications. Annu Rev Biomed Eng 10:247–274. https://doi.org/10.1146/annurev.bioeng.10.061807.160547
32. Hulzink J, Konijnenburg M, Ashouei M, Breeschoten A, Berset T, Huisken J, Stuyt J, de Groot H, Barat F, David J, Van Ginderdeuren J (2011) An ultra low energy biomedical signal processing system operating at near-threshold. IEEE Trans Biomed Circuits Syst 5(6):546–554. https://doi.org/10.1109/TBCAS.2011.2176726
33. Konijnenburg M, Cho Y, Ashouei M, Gemmeke T, Kim C, Hulzink J, Stuyt J, Jung M, Huisken J, Ryu S, Kim J (2013) Reliable and Energy-Efficient 1MHz 0.4V Dynamically Reconfigurable SoC for ExG applications in 40nm LP CMOS. In 2013 IEEE international solid-state circuits conference digest of technical papers, pp 430–431. https://doi.org/10.1109/ISSCC.2013.6487801