I J C T A, 9(34) 2016, pp. 381-390
© International Science Press

An FPGA-based Implementation of an Efficient Vector Coprocessor using Reconfigurable FSM for Vector Detection based Control

Nitish Das[1] and Aruna Priya P.[1]

ABSTRACT

Nowadays, the vector coprocessor (VP) is taken as a potential solution for multimedia data processing and control applications. Due to partial re-programmability and hardware sharing, field programmable gate array (FPGA) offers a novel platform to implement it. The prime objective of this study stems from creating an efficient VP architecture for vector detection. The reconfigurable finite state machine (FSM) based structures are investigated to develop such techniques. A reconfigurable FSM-based architecture is designed for simultaneous detection of m-bit long n-vectors within a bit-stream. A particular state encoding technique is developed to reduce the chip area and to enhance the processing speed for the proposed architecture. A complete mathematical foundation is made for defining the state transitions and output function in such FSM. The proposed techniques introduce a solution to overcome the issue of operating with different length vectors in VPs as they provide flexibility in two aspects (i.e. m and n).

Keywords: Field programmable gate array (FPGA), reconfigurable finite state machine, state encoding technique, vector coprocessor, vector detection based controller.

I. INTRODUCTION

Vector coprocessors (VPs) have proved their significance in the field of multimedia data processing [1]-[2]. VPs are investigated in the applications, which involve sequence matching and MINMAX operations, such as digital image processing, pattern recognition [3], network intrusion detection [4] and fuzzy control [5], [11]. An FPGA-based platform i.e. Xilinx ZYNQ 7000 ZC702 System-on-Chip (SoC) is considered for implementation of the applications as mentioned earlier. FPGAs come with an advantage of partial programmability and hardware sharing during the run-time execution of an application, which offers high processing speed and reduced power consumption [6]-[7].

To develop such applications using VPs mainly involve the simultaneous detection of input vectors within a serialized bit stream generated by the central processor unit and to produce outputs associated with it [9]. This issue becomes tedious when input vector set and its output matrix become fuzzy (i.e. always changing over time) [5].

An accurate, stable and reconfigurable design is required to solve the problem of vector detection in VPs. An efficient utilization of hardware is also needed to make it viable. Therefore, the reconfigurable FSM model is considered to design it [12]. In a reconfigurable FSM, the states are configured to perform a particular operation [13].

FSMs furnish a beautiful framework to solve control problems, which involves a sequential operation [14]. Unlike regular sequential circuits, the state transition of an FSM is more complicated. The design of

[1] Dept. of ECE, SRM University, Kattankulathur- 603203, Chennai, India, *E-mail: nitishdas99@gmail.com*

FSM typically starts with an abstract graphic description, such as a state diagram. A state diagram consists of nodes and one-directional transition arcs. A node symbolizes a unique state of the FSM, and an arc signifies a transition from one state to another [15].

An FSM-based detection of desired vectors is discussed using binary search tree technique and shown in Fig. 1. (i) The desired vectors should be arranged in ascending or descending order. (ii) According to the occurrence of switching, a column-wise separation and partition of bits are performed. It is similar to creating a unique input/output tree [16]. (iii) Default states for input 0 (if a state is not defined for 0) and 1 (if a state is not defined for 1) are defined. Then, the state diagram is formed. During simultaneous detection of multiple vectors, the occurrence of overlapping of vectors leads the FSM to go in an unpredictable state, which results in failure [14]. This problem is solved by resetting the FSM at every m^{th} clock cycle. All the desired vectors are detected only in the states present in the last column, at the occurrence of a particular input.

The creation of a reconfigurable state of an FSM for a vector detection necessitates, selecting the required states, input-output combinations and possible transitions between them [12]-[13]. It results in a state-space explosion. The computations required defining a reconfigurable state, increase with the growing complexity of a state diagram, because state reconfiguration depends on certain conditions and inputs [13]. The mentioned complexity upsurges as per increase in (n, m) values. Furthermore, data dependent control specifications become monotonous to be expressed through FSM at abstraction level. A particular state encoding technique is developed to maintain design tradeoffs such as logical depth, chip area and power consumption for this model [15], [17]. During sequential operation of an FSM, temporal transient faults (soft errors) lead system to go in an unpredictable state. The concept of state protection [18] by replicating the frequently visited states has been implemented to make the system more reliable.

The proposed technique requires less number of embedded memory blocks (EMBs) [19] for implementation using FPGAs. The proposed method also provides a complete framework to design adaptive control systems, as it provides adaptability in three aspects i.e. the length of the vector, the number of vectors and precedence based output selection.

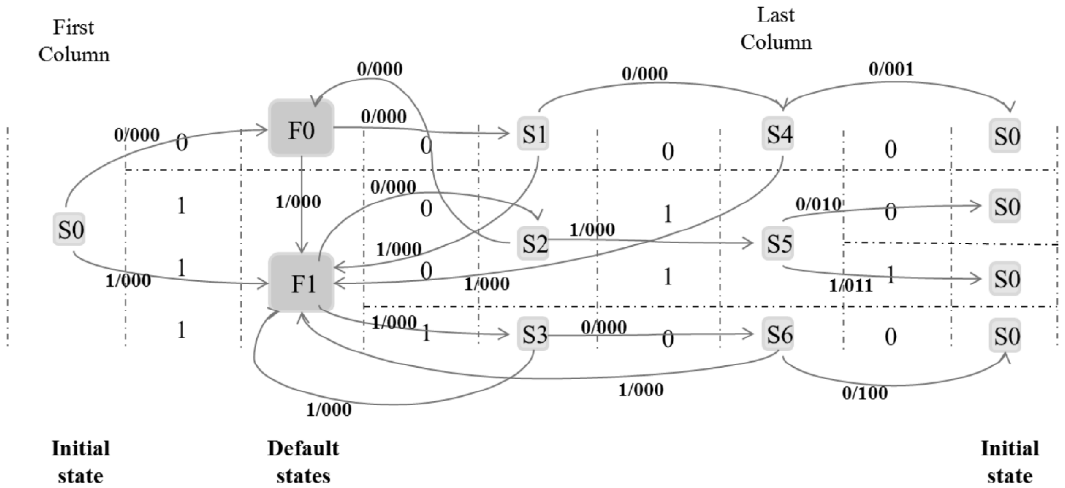


Figure 1: FSM for detection of 4-bit long constant vectors 0000, 1100, 1011 and 1010

II. PROPOSED ARCHITECTURE

The proposed VP architecture for vector detection consists of three fundamental blocks as shown in Fig. 2.

The crossbar switch is used as in [10]. It arranges m-bit long n-vectors vectors in ascending order i.e. $v1, v2, v3, \ldots\ldots, vn$ with its corresponding k-bit long outputs i.e. $r[1], r[2], r[3], \ldots\ldots, r[n]$. Priority is assigned based on the weight (i.e. value) of the vector while designing the crossbar switch. It is also considered that a higher value of the vector leads to more precedence. So, the output vectors of crossbar switch i.e. $da[1], da[2], da[3],\ldots\ldots, da[n\text{-}1], da[n]$ are arranged in ascending order.

The reconfigurable FSM is used to detect desired n-vectors of m-bit length which are arranged in ascending order. It produces user defined output combinations i.e. *out[1], out[2],....., out[k]* on detection of desired vectors from the input bit-stream.

An m-bit counter is used in the proposed architecture for the generation of a reset signal at every m^th^ clock period. In this way, the problem of overlapping of vectors is overcome while simultaneous detection of multiple vectors.

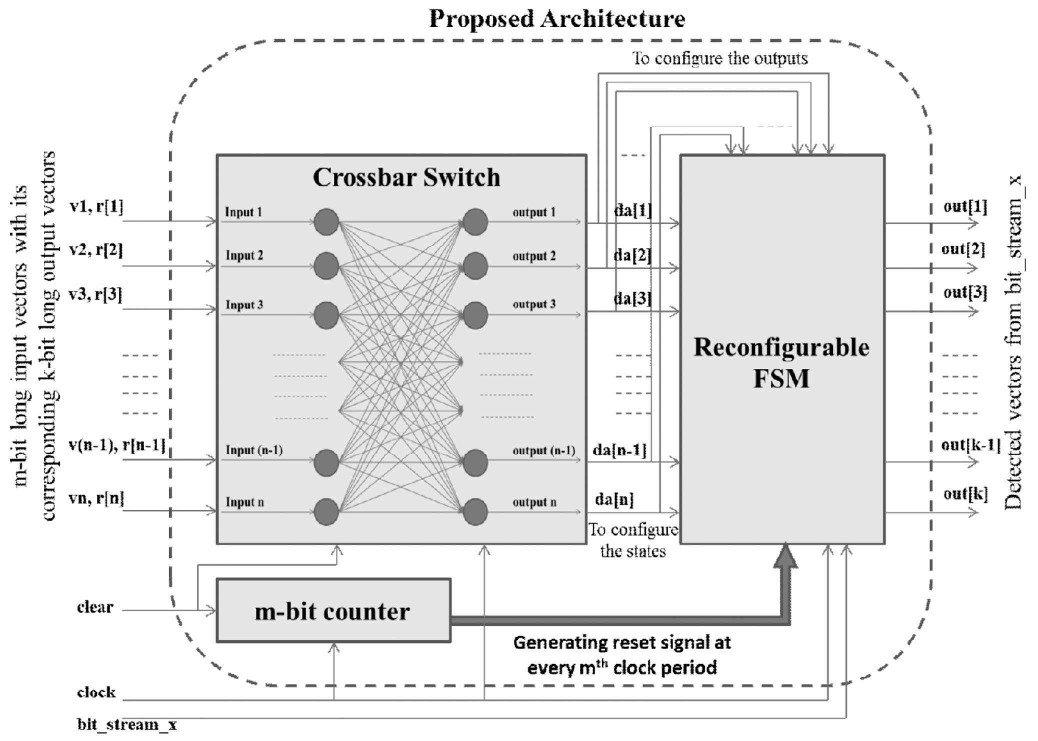


Figure 2: Proposed VP architecture using reconfigurable FSM for vector detection

The proposed reconfigurable FSM model for detection of m-bit long n-vectors i.e. $da[1], da[2], da[3],\ldots\ldots, da[n\text{-}1], da[n]$ from a serialized bit-stream is presented in Fig. 3. Variable $x[n\text{-}1][m\text{-}1]$ is used to detect the switching (i.e. 0 to 1 or vice versa) between two adjacent vectors. Variable $z[n][m]$ is used to compare a particular bit value of the vector w.r.t. 0.

A particular state encoding technique is developed, in which two types of states have been considered such as (i) conventional state and (ii) Initial and default states [17, 18]. States present only in the first and second column are considered as the initial and default states. Rest of the states is regarded as conventional states.

A conventional state consists of two parts i.e. configure bits ($\lceil \log_2 n \rceil = k$) and difference bits ($\lceil \log_2 (m-2) \rceil = g$). So, the required number of state encoding bits to define a conventional state is obtained by Eq. (1).

$$\begin{aligned} &\text{Required number of state encoding bits},\ p \\ &= \lceil \log_2 n \rceil + \lceil \log_2 (m\text{-}2) \rceil + 2 \end{aligned} \tag{1}$$

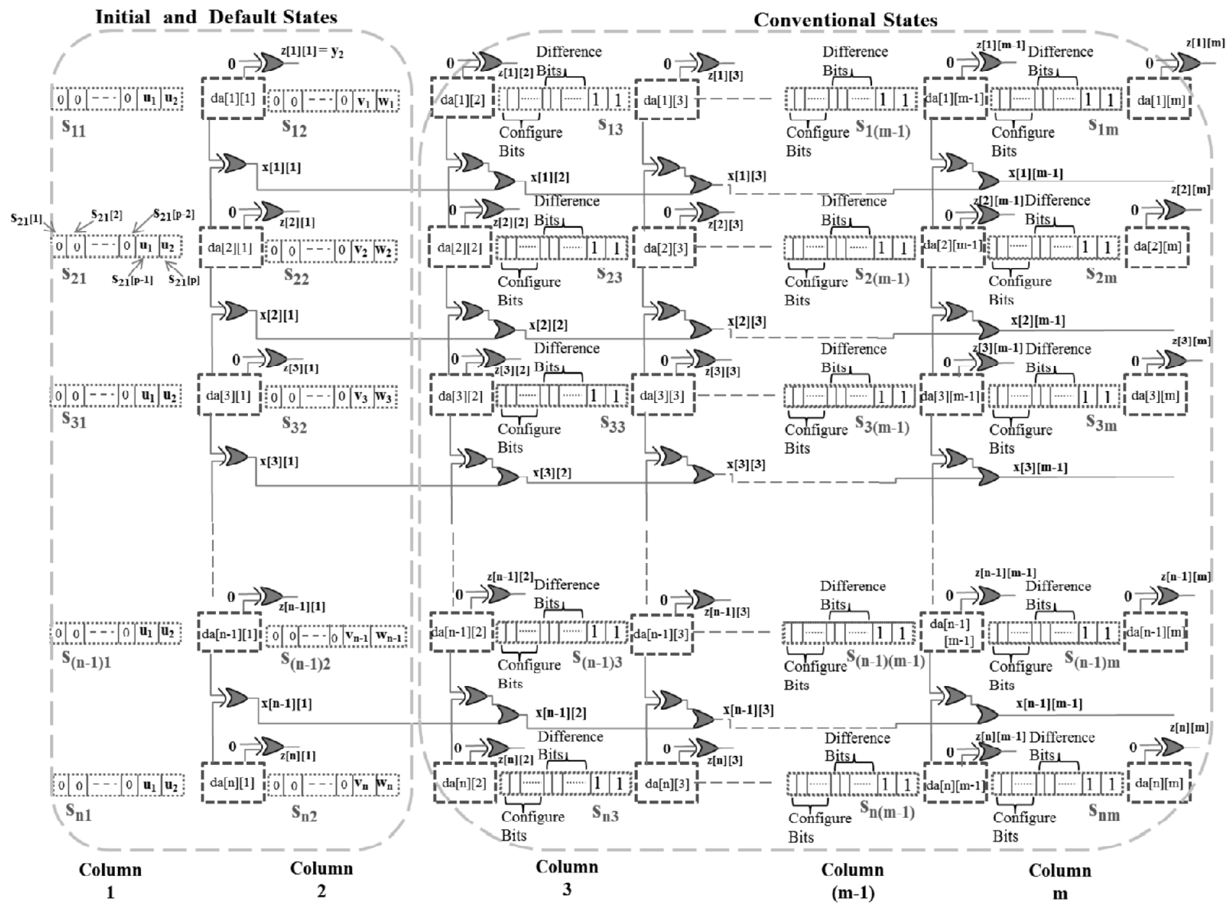


Figure 3: Proposed reconfigurable FSM model for detection of m-bit long n-vectors

The configure bits are used to set up the states of the FSM row-wise in the direction top to bottom in a column. Configure bits are assigned depending upon the row-wise switching between two adjacent vectors. Configure bits for the leading state in a column (i.e. s_{1m}) is taken as $\begin{bmatrix}0 & 0 & 0 & ..... & 0\end{bmatrix}_{(1\times k)}$. Let the configure bits of the state s_{nm} are $c_1,\ c_2,\ c_3,\ ....,c_k$ and the state $s_{(n\text{-}1)m}$ are $c'_1,\ c'_2,\ c'_3,\ ......,c'_k$. So, the relations between configure bits of state s_{nm} and $s_{(n\text{-}1)m}$ are defined by Eq. (2).

$$\begin{bmatrix} c_1\\ c_2\\ c_3\\ \dots\\ \dots\\ \dots\\ c_k \end{bmatrix}_{k\times 1} = \overline{x[n-1][m]} \bullet \begin{bmatrix} c'_1\\ c'_2\\ c'_3\\ \dots\\ \dots\\ \dots\\ c'_k \end{bmatrix}_{k\times 1} + x[n-1][m] \bullet \left[binary\ equivalent\ of\ (n-1)\right]^T_{1\times k} \tag{2}$$

For proper operation of the proposed detector, any two states of a different column can never be identical. Difference bits are introduced to confirm this particular condition. It creates a distinction between the states of the FSM column-wise in the direction MSB to LSB. Let the difference bits of the state s_{nm} are $df_1,\ df_2,\ df_3,.....,\ df_g$. So, the difference bits of the state s_{nm} are defined by Eq. (3).

$$\begin{bmatrix} df_1 \\ df_2 \\ df_3 \\ \cdots \\ \cdots \\ \cdots \\ df_g \end{bmatrix}_{g\times 1} = [binary\ equ\ ivalent\ of\ (m\text{-}3)]^T_{1\times g} \tag{3}$$

For all conventional states, the last two bits present in the LSB should be 1 because they are preserved for defining the initial and default states.

The following set of objectives has been formed to define initial and default states for the states present in the first and second column:

1. All the states present in column 1 should act as the initial state.
2. The default state for input 0 (i.e. states which are not defined for input 0) should always be $[0 \quad 0 \quad .... \quad 0 \quad 0 \quad 0 \quad 1]_{1\times p}$.
3. The default state for input 1 (i.e. states which are not defined for input 1) should always be $[0 \quad 0 \quad .... \quad 0 \quad 0 \quad 1 \quad 0]_{1\times p}$.
4. The initial and default states are the frequently visited states. So, they should be replicated to avoid soft errors [18].

The objectives as mentioned earlier are obtained by combined operation of Eq. (4), Eq. (5), Eq. (6), Eq. (7) and Eq. (8).

$$y_1 = x[1][1] + x[2][1] + x[3][1] + ... + x[n\text{-}1][1] \tag{4}$$

$$u_1 = y_1 \bullet y_2 \tag{5}$$

$$u_2 = y_1 \bullet \overline{y_2} \tag{6}$$

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \cdots \\ \cdots \\ \cdots \\ v_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} 0 & 0 & 0 & ...... & 0 \\ 1 & 0 & 0 & ...... & 0 \\ 1 & 1 & 0 & ...... & 0 \\ .. & .. & .. & ...... & .. \\ .. & .. & .. & ...... & .. \\ .. & .. & .. & ...... & .. \\ 1 & 1 & 1 & ...... & 1 \end{bmatrix}_{n\times(n-1)} \times \begin{bmatrix} x[1][1] \\ x[2][1] \\ x[3][1] \\ .... \\ .... \\ .... \\ x[n\text{-}1][1] \end{bmatrix}_{(n-1)\times 1} + u_2 \tag{7}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \cdots \\ \cdots \\ \cdots \\ w_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} 1 & 1 & 1 & ...... & 1 \\ 0 & 1 & 1 & ...... & 1 \\ 0 & 0 & 1 & ...... & 1 \\ .. & .. & .. & ...... & .. \\ .. & .. & .. & ...... & .. \\ .. & .. & .. & ...... & .. \\ 0 & 0 & 0 & ...... & 0 \end{bmatrix}_{n\times(n-1)} \times \begin{bmatrix} x[1][1] \\ x[2][1] \\ x[3][1] \\ .... \\ .... \\ .... \\ x[n\text{-}1][1] \end{bmatrix}_{(n-1)\times 1} + u_1 \tag{8}$$

For all the state, present in column 1, the next state function for input 0 (i.e. $n0s_{nm}$) and input 1 (i.e. $n1s_{nm}$) is defined by $\begin{bmatrix}0 & 0 & .... & 0 & 0 & 0 & 1\end{bmatrix}_{1\times p}$ and $\begin{bmatrix}0 & 0 & .... & 0 & 0 & 1 & 0\end{bmatrix}_{1\times p}$ respectively.

For all the state, present within the column 2 and the column (m-1), the next state function for input 0 and input 1 is given by Eq. (9) and Eq. (10) respectively. The idea lies in updating the next state function as per the occurrence of switching between two adjacent vectors row-wise. This updating process is performed in the direction top to bottom (in the case of input 0) and bottom to top (in the case of input 1).

For states, located in the mth column, the next state function for input 0 and input 1 can be either a default state or an initial state. Hence, for these states, the next state function is given by Eq. (11) and Eq. (12) respectively. Here, any non-existing value of $x[n][m]$ is considered to be 1.

$$\begin{bmatrix} n0s_{nm}[1] \\ n0s_{nm}[2] \\ n0s_{nm}[3] \\ ..... \\ ..... \\ ..... \\ n0s_{nm}[p-2] \\ n0s_{nm}[p-1] \\ n0s_{nm}[p] \end{bmatrix}_{p\times 1} = z[n][m] \bullet \begin{bmatrix} s_{n(m+1)}[1] \\ s_{n(m+1)}[2] \\ s_{n(m+1)}[3] \\ ...... \\ ...... \\ ...... \\ s_{n(m+1)}[p-2] \\ s_{n(m+1)}[p-1] \\ s_{n(m+1)}[p] \big/ z[n][m] \end{bmatrix}_{p\times 1} + \overline{x[n-1][m-1]} \bullet \begin{bmatrix} n0s_{(n-1)m}[1] \\ n0s_{(n-1)m}[2] \\ n0s_{(n-1)m}[3] \\ ..... \\ ..... \\ ..... \\ n0s_{(n-1)m}[p-2] \\ n0s_{(n-1)m}[p-1] \\ 0 \end{bmatrix}_{p\times 1} \tag{9}$$

$$\begin{bmatrix} n1s_{nm}[1] \\ n1s_{nm}[2] \\ n1s_{nm}[3] \\ ..... \\ ..... \\ ..... \\ n1s_{nm}[p-2] \\ n1s_{nm}[p-1] \\ n1s_{nm}[p] \end{bmatrix}_{p\times 1} = \overline{z[n][m]} \bullet \begin{bmatrix} s_{n(m+1)}[1] \\ s_{n(m+1)}[2] \\ s_{n(m+1)}[3] \\ ...... \\ ...... \\ ...... \\ s_{n(m+1)}[p-2] \\ s_{n(m+1)}[p-1] \big/ \overline{z[n][m]} \\ s_{n(m+1)}[p] \end{bmatrix}_{p\times 1} + \overline{x[n+1][m-1]} \bullet \begin{bmatrix} n1s_{(n+1)m}[1] \\ n1s_{(n+1)m}[2] \\ n1s_{(n+1)m}[3] \\ ..... \\ ..... \\ ..... \\ n1s_{(n+1)m}[p-2] \\ 0 \\ n1s_{(n+1)m}[p] \end{bmatrix}_{p\times 1} \tag{10}$$

$$\begin{bmatrix} n0s_{nm}[1] \\ n0s_{nm}[2] \\ n0s_{nm}[3] \\ ..... \\ ..... \\ ..... \\ n0s_{nm}[p-2] \\ n0s_{nm}[p-1] \\ n0s_{nm}[p] \end{bmatrix}_{p\times 1} = x[n-1][m-1] \bullet \begin{bmatrix} 0 \\ 0 \\ 0 \\ ...... \\ ...... \\ ...... \\ 0 \\ s_{n1}[p-1] \bullet z[n][m] \\ s_{n1}[p] + \overline{z[n][m]} \end{bmatrix}_{p\times 1} + \overline{x[n-1][m-1]} \bullet \begin{bmatrix} 0 \\ 0 \\ 0 \\ ..... \\ ..... \\ ..... \\ 0 \\ n0s_{(n-1)m}[p-1] \\ n0s_{(n-1)m}[p] \end{bmatrix}_{p\times 1} \qquad (11)$$

$$\begin{bmatrix} n1s_{nm}[1] \\ n1s_{nm}[2] \\ n1s_{nm}[3] \\ ..... \\ ..... \\ ..... \\ n1s_{nm}[p-2] \\ n1s_{nm}[p-1] \\ n1s_{nm}[p] \end{bmatrix}_{p\times 1} = x[n+1][m-1] \bullet \begin{bmatrix} 0 \\ 0 \\ 0 \\ ...... \\ ...... \\ ...... \\ 0 \\ s_{n1}[p-1] + z[n][m] \\ s_{n1}[p] \bullet \overline{z[n][m]} \end{bmatrix}_{p\times 1} + \overline{x[n+1][m-1]} \bullet \begin{bmatrix} 0 \\ 0 \\ 0 \\ ..... \\ ..... \\ ..... \\ 0 \\ n1s_{(n+1)m}[p-1] \\ n1s_{(n+1)m}[p] \end{bmatrix}_{p\times 1} \qquad (12)$$

All outputs of the proposed detector are defined for the states, present in the mth column at the arrival of the unique input. The output vectors are considered to be zero for the rest of the states.

Let $r[1], r[2], r[3],......., r[n]$ are the output vectors associated with the vectors $v1, v2, v3,......., vn$ respectively. The output vectors are either user defined or can be defined using precedence among vectors (i.e. $da[1],\ da[2],\ da[3],..........,\ da[n\text{-}1],\ da[n]$). Each output vector consists of $\lceil \log_2 n \rceil = k$ bits to maintain design tradeoffs. One k-bit long variable $out[k]$ (i.e. output function) is used for the representation of the output vector during vector detection. Hence, the output function is defined by Eq. (13).

$$\begin{bmatrix} out[1] \\ out[2] \\ out[3] \\ \cdots\cdots \\ \cdots\cdots \\ out[k] \end{bmatrix}_{k\times 1} = x \bullet z[n][m] \bullet \begin{bmatrix} r[n][1] \\ r[n][2] \\ r[n][3] \\ \cdots\cdots \\ \cdots\cdots \\ r[n][k] \end{bmatrix}_{k\times 1} + \bar{x} \bullet \overline{z[n][m]} \bullet \begin{bmatrix} r[n][1] \\ r[n][2] \\ r[n][3] \\ \cdots\cdots \\ \cdots\cdots \\ r[n][k] \end{bmatrix}_{k\times 1} \quad (13)$$

III. EXPERIMENTAL RESULTS

Experiments have been conducted to observe the performance of the proposed system for the various combination of (m, n). The simulation results have been obtained using Xilinx ISE Design Suite as shown in Fig. 5 (Note: All variable contains their usual meaning). Hardware implementations have been performed for the proposed system using Xilinx ZYNQ 7000 ZC702 System-on-Chip (SoC); which validates the simulation results. Various parameters regarding the hardware implementation of the proposed system with its percentage usage have been presented in Table I. Static power dissipation is found to be 0.081W (for the typical process) and 0.136W (for the commercial process) for the various combination of (m, n).

The proposed vector detection method is investigated for image watermarking. In this experiment, 1 bit/pixel binary watermark of size (64 x 64) and 8bits/pixel gray scale cover of size (256 x 256) is considered as in [20]. The comparison results are shown in Fig. 4. Here, the cover image is partitioned into (4 x 8) blocks i.e. proposed VP with m=8 and n=4 is considered. The watermark image is partitioned into (2 x 2) i.e. proposed VP with m=2 and n=2 is considered. MATLAB is used for conversion of integer image data to binary. Implementation results show that the proposed VP requires only 10% of memory elements as compared with [20]. But, due to a number of feedback loops used in the FSM, latency acts as a tradeoff for it. Consequently, the maximum operating frequency is found to be 20.78 MHz.

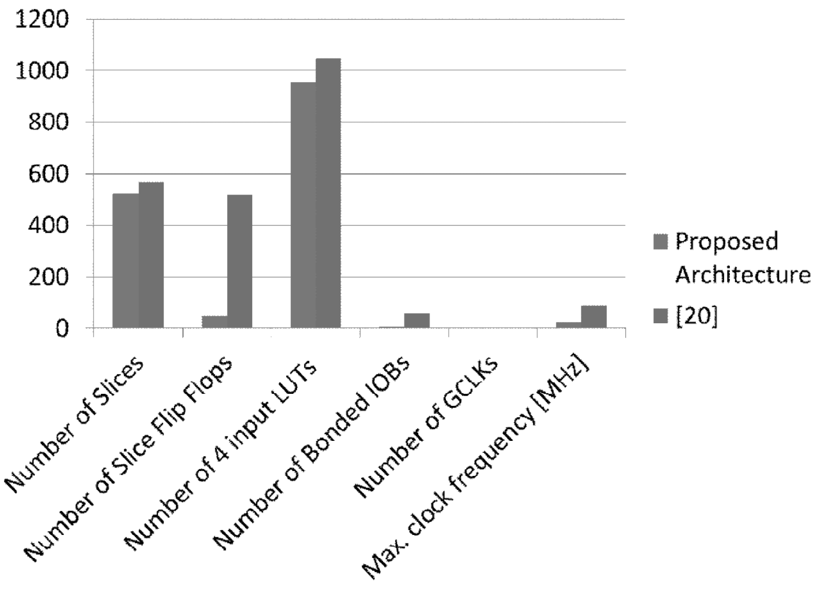


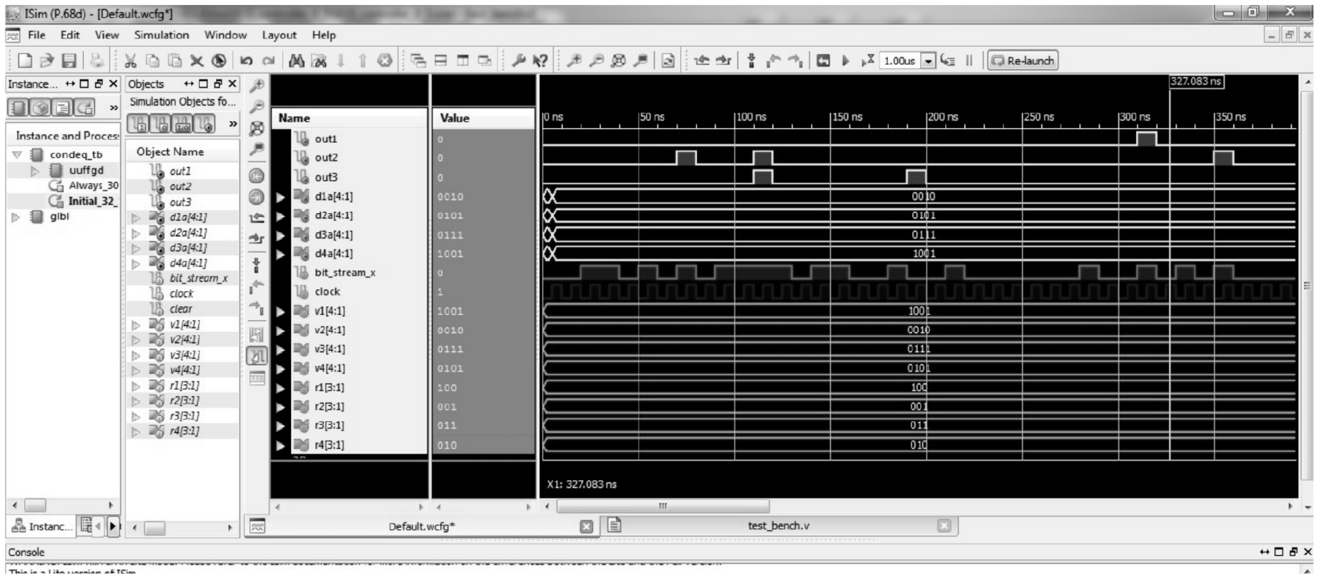
Figure 4: Performance comparison based on hardware implementation

IV. CONCLUSION

A design technique to detect m-bit long n-vectors using a reconfigurable FSM-based vector coprocessor (VP) is proposed. The proposed vector detection method offers reduced consumption of embedded memory blocks (EMBs) for its operation. It provides 90% reduction in EMBs requirement as compared with [20] in image watermarking applications. It also makes the system, tolerant to soft errors because state protection [18] is introduced in the proposed state encoding scheme. The hardware requirement (i.e. LUT consumption) to implement the proposed system varies approximately by 14% and 40% as per increase in the value of m and n respectively. The proposed techniques can be used to create VPs with a reduced hardware using reconfigurable processing fabric (RPF) for applications, such as image processing, word spotting [3] and network intrusion detection.

Table I
Performance of the proposed system for the various combination of (m, n)

S. No.	*(m, n)*	*Number of 4-input LUTs*	*Number of occupyed Slices*	*Number of Sclice Flip Flops*	*Number of bonded IOBs*	*Maximum Operating Frequency (MHz)*	*Maximum Path Delay (ns)*	*Memory Usage (MB)*	*Number of BUFG MUXs*	*Average Fanout of Non-clock Nets*
1	(4, 3)	212 (2%)	122 (2%)	21 (1%)	30 (12%)	30.598	6.916	212.804	1 (4%)	3.92
2	(4, 4)	389 (4%)	215 (4%)	25 (1%)	38 (16%)	23.327	8.36	216.9	1 (4%)	3.82
3	(4, 5)	968 (10%)	513 (11%)	34 (1%)	46 (19%)	20.161	9.569	228.164	1 (4%)	4.14
4	(5, 3)	273 (2%)	157 (3%)	26 (1%)	36 (15%)	29.63	6.92	214.852	1 (4%)	3.82
5	(5, 4)	524 (5%)	289 (6%)	31 (1%)	46 (19%)	22.742	8.715	221.572	1 (4%)	3.88
6	(5, 5)	1160 (12%)	631 (13%)	53 (1%)	56 (24%)	19.842	11.836	268.036	1 (4%)	4.2
7	(6, 3)	373 (4%)	212 (4%)	30 (1%)	42 (18%)	29.296	8.193	216.9	1 (4%)	3.9
8	(6, 4)	681 (7%)	375 (8%)	36 (1%)	54 (23%)	22.667	8.112	224.772	1 (4%)	3.85
9	(6, 5)	1336 (14%)	723 (15%)	53 (1%)	66 (28%)	18.797	9.273	271.876	1 (4%)	4.17
10	(7, 3)	448 (4%)	254 (5%)	35 (1%)	48 (20%)	26.991	8.13	218.948	1 (4%)	3.67
11	(7, 4)	831 (8%)	457 (9%)	42 (1%)	62 (26%)	20.961	10.758	229.252	1 (4%)	3.73
12	(7, 5)	1506 (16%)	809 (17%)	56 (1%)	76 (32%)	18.039	8.293	274.948	1 (4%)	4.1
13	(8, 3)	539 (5%)	302 (6%)	39 (1%)	54 (23%)	25.983	8.696	222.596	1 (4%)	3.79
14	(8, 4)	955 (10%)	524 (11%)	47 (1%)	70 (30%)	20.78	9.576	263.684	1 (4%)	3.75
15	(8, 5)	1735 (18%)	939 (20%)	71 (1%)	86 (37%)	18.33	10.049	286.532	1 (4%)	4.13



Name	Value
out1	0
out2	0
out3	0
d1a[4:1]	0010
d2a[4:1]	0101
d3a[4:1]	0111
d4a[4:1]	1001
bit_stream_x	0
clock	1
v1[4:1]	1001
v2[4:1]	0010
v3[4:1]	0111
v4[4:1]	0101
r1[3:1]	100
r2[3:1]	001
r3[3:1]	011
r4[3:1]	010

Figure 5: Simulation results for the proposed system in the case of (4, 4)

REFERENCES

[1] S. F. Beldianu, C. Dahlberg, T. Steele, S. G. Ziavras, Versatile design of shared vector coprocessors for multicores, (2012) *Microprocess. Microsyst.*, 36 (7), pp. 543–554.

[2] S. F. Beldianu, S. G. Ziavras, Multicore-Based Vector Coprocessor Sharing for Performance and Energy Gains, (2013) *ACM Trans. Embed. Comput. Syst.*, 13 (2), pp. 1-25.

[3] T. Mondal, N. Ragot, J. Ramel, U. Pal, Flexible Sequence Matching technique/ : An effective learning-free approach for word spotting, (2016) *Pattern Recognit.*, 60, pp. 596–612.

[4] L. Qiyue, L. Jie, W. Jianping, Z. Baohua, Q. Yugui, A pipelined processor architecture for regular expression string matching, (2012) *Microprocess. Microsyst.*, 36, pp. 520–526.

[5] T. Hollstein, S. K. Halgamuge, M. Glesner, Computer-Aided Design of Fuzzy Systems Based on Generic VHDL Specifications, (1996) *IEEE Trans. Fuzzy Syst.*, 4 (4), pp. 403-417.

[6] A. Al-Wattar, S. Areibi, G. Grewal, An Efficient Evolutionary Task Scheduling / Binding Framework for Reconfigurable Systems, (2016) *Int. J. Reconfigurable Comput.*, 2016, pp. 1–24.

[7] F. A. Hussin, Z. E. M. Osman, L. Xia, N. B. Z. Ali, Optimization of Processor Architecture for Sobel Real-Time Edge Detection Using FPGA, (2013) *International Review on Computers and Software (IRECOS)*, 8 (4), pp. 970-982.

[8] E. Martins, L. Almeida, A. Fonseca, An FPGA-based coprocessor for real-time fieldbus traffic scheduling — architecture and implementation, (2004*) J. Syst. Archit.*,51, pp. 29–44.

[9] Y. Lu, S. Rooholamin, S. G. Ziavras, Vector Coprocessor Virtualization for Simultaneous Multithreading, (2016) *ACM Trans. Embed. Comput. Syst.*,15 (3), pp. 1-25.

[10] Zhang, X., Mohanty, S. R., Bhuyan, L. N., Adaptive Max-min Fair Scheduling in Buffered Crossbar Switches Without Speedup, *Proceedings of the 26th IEEE International Conference on Computer Communications INFOCOM 2007* (Page: 454 Year of Publication: 2007 ISBN:1-4244-1047-9).

[11] M. J. Patyra, J. L. Grantner, K. Koster, Digital Fuzzy Logic Controller: Design and Implementation, (1996) *IEEE Trans. Fuzzy Syst.*, 4 (4), pp. 439-459.

[12] V. Sklyarov, Reconfigurable models of finite state machines and their implementation in FPGAs, (2002) *J. Syst. Archit.*, 47, pp. 1043-1064.

[13] J. Glaser, M. Damm, J. Haase, C. Grimm, TR-FSM: Transition-Based Reconfigurable Finite State Machine, (2011) *ACM Trans. Reconfigurable Technol. Syst.*, 4 (3), pp. 1-14.

[14] I. Ahmad, F. M. Ali, A. S. Das, Synthesis of finite state machines for improved state verification, (2006) *Comput. Electr. Eng.*, 32 (5), pp. 349-363.

[15] R. Czerwinski, D. Kania, Area and speed oriented synthesis of FSMs for PAL-based CPLDs, (2011) *Microprocess. Microsyst.*, 36, pp. 45-61.

[16] K. Naik, Efficient computation of unique input/output sequences in finite-state machines, (1997) *IEEE/ACM Trans. Netw.*, 5 (4), pp. 585-599.

[17] A. H. El-Maleh, S. M. Sait, A. Bala, State assignment for area minimization of sequential circuits based on cuckoo search optimization, (2015) *Comput. Electr. Eng.*, 44, pp. 13-23.

[18] A. H. El-maleh, A. S. Al-qahtani, A finite state machine based fault tolerance technique for sequential circuits, (2014) *Microelectron. Reliab.*, 54, pp. 654-661.

[19] M. Kolopienczyk, A. Barkalov, L. Titarenko, M. Wegrzyn, Hardware reduction for EMB-based Mealy FSM, (2015) in IFAC- PapersOnLine, 48 (10), pp. 202-207.

[20] S. Ghosh, S. Talapatra, J. Sharma, N. Chatterjee, H. Rahaman, S. P. Maity, Dual Mode VLSI Architecture for Spread Spectrum Image Watermarking using Binary Watermark, (2012) *Procedia Technology*, 6, pp. 784–791.