I J C T A, 9(24), 2016, pp. 363-373 ISSN: 0974-5572

An FPGA-based implementation of a vector coprocessor for control applications

Nitish Das* and P. Aruna Priya**

ABSTRACT

The vector coprocessor (VP) is taken as a potential solution for control applications because of its low power consumption and internal feedback network. Due to partial re-programmability and hardware sharing, field programmable gate array (FPGA) offers a novel platform to implement it. Architecture is presented for simultaneous detection of the desired non-overlap n-binary control vectors of a constant length of m-bit with reduced hardware requirement. Finite state machine (FSM) has been investigated for designing the proposed detector. Within a bit stream, which is generated by central processor unit (CPU), it provides a single output or user defined output combinations for desired control vectors. The reduced state table for its FSM implementation is obtained without using state diagram and state reduction techniques. A formula has been developed for calculation of an approximate number of memory elements required for implementation of the proposed detector. One of its application is also conferred to add new devices to any existing digital system or to modify its features. It provides adaptability in three aspects i.e. the length of sequences, the number of sequences and user defined output combinations.

Keywords: Field programmable gate array (FPGA), finite state machine, control vector detector, state diagram, state reduction, vector coprocessor (VP).

1. INTRODUCTION

Vector coprocessors (VPs) have proved their significance in the field of multimedia data processing [20]. An FPGA-based platform (i.e. Spartan 3E Starter Board) is considered for its hardware implementation. The motivation to develop control applications using VPs stems from simultaneous detection of input control vectors within a serialized bit stream generated by the central processor unit and to produce outputs associated with it.

This paper presents a new designing technique that generates an optimal set of states to implement an FSM design for simultaneous detection of multiple control vectors of a constant length. A detector is required to identify the advent of a particular m-bit length binary control vector within a serialized bit stream. It provides the user defined outputs upon the arrival of the last bit of the desired binary control vector [1].

The prime objective of this study concerns about creating an accurate, stable and flexible design for controlling operations as well as to keep it economical. Therefore, an efficient utilization of hardware is required to make the design cost effective. It can be obtained by the FSM model which is a circuit with the internal states [2], [3]. FSMs furnish a beautiful framework to solve control problems, which involves a sequential operation [4].

Unlike regular sequential circuits, the state transition of an FSM is more complicated. The design of FSM typically starts with an abstract graphic description, such as a state diagram [5], [6]. A state diagram consists of nodes and one-directional transition arcs [7]. A node symbolizes a unique state of the FSM, and an arc signifies a transition from one state to another. The creation of a state diagram for the proposed

* Dept. of ECE, SRM University, Kattankulathur-603203, Chennai, India, *Email: nitishdas99@gmail.com*

** Dept. of ECE, SRM University, Kattankulathur-603203, Chennai, India.

detector necessitates, selecting the required states, input-output combinations and possible transitions between them, which results in state-space explosion. The corresponding state table is formed by the state diagram.

From state table, a reduced state table is created by using the merger graph method [8] or other state reduction techniques [2], [3], which results in the saving of precious memory. An efficient state encoding technique [6], [9] is used to reduce the power consumption further during synthesis. The computation required obtaining a reduced state table increases with the growing complexity of a state diagram. The mentioned complexity upsurges as per increase in (n, m) values. Furthermore, data dependent control specifications become tedious to be expressed through FSM at abstraction level.

The proposed algorithm overcomes the problem of forming a state diagram for the control vector detector. It provides the simplest way to obtain a corresponding reduced state table without utilization of monotonous computation. While creating prototypes using FPGA [10], [11], the proposed method can develop efficient techniques for implementation of FSMs using less number of embedded memory blocks (EMBs) [12], [13]. The proposed method provides a complete framework to design adaptive control systems [14].

Device portability is termed as, adding new devices to the port or changing the configuration of the existing output device connected to that port. Nowadays, it is getting attention, while designing modern systems. It creates a new era of developing flexible systems, which supports the addition of new hardware and reconfiguration in its features. It offers promising results for robotics and its applications. But the potential problems lie in controllability [15] of such devices. A possible solution has been introduced as an application of the proposed detector i.e. a new approach for controlling the devices by control vector matching.

2. PROPOSED ARCHITECTURE

The proposed detector consists of two parts as shown in Figure 1. The part-1 is used to detect desired n-binary control vectors of m-bit length by a single output or user defined output combinations. The part-2 is used to identify all n-binary control vectors of m-bit length by an output 1. The incapability in the detection of overlapped control vectors is the primary disadvantage of the proposed algorithm. Overlapping of control vectors will occur, if the proposed algorithm is applied to detect multiple control vectors simultaneously, which results in failure. A solution is introduced as the part-2 of the architecture, which is used for generation of a reset signal to overcome the mentioned problem. The output of the part-2 is used as a reset signal (i.e. signal for moving all states to the initial state in the FSM present in the part-1).

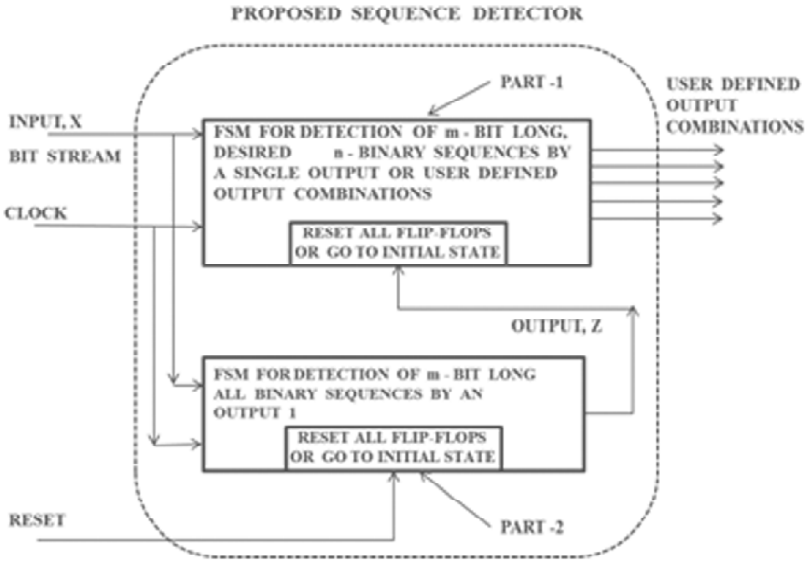

Figure 1: Proposed control vector detector architecture.

2.1. Control Vector Detection by a Single Output combination:

To illustrate all aspects and conditions and to compare its performance with [1] examples have been introduced.

Example 1. Let us consider the part-1 of the detector will produce a single output combination i.e. 1, when any one of the following non-overlap binary control vector of a constant length of 21 bit is detected: 100011001000111110000, 100011001010111111000, 100011001000111110011, 100011001010111111011, 100011001000111110110, 100011001000111110111, 100011001010111111110, 100011001010111111111 and the part-2 of the detector will produce output 1, when any 21-bit long sequence is detected. To obtain an optimal hardware for the proposed detector the following designing steps [13] should be followed:

2.1.1. Step (1): Arranging the desired sequences in ascending or descending order

First of all, the desired vectors should be arranged in ascending or descending order according to its weight.

2.1.2. Step (2): According to the occurrence of switching, making a column-wise separation and partition of bits

Now, the separation of bits of the desired vectors from each other should be performed to form columns. After that, a column-wise search for switching i.e. 1 to 0 or 0 to 1 should be accomplished. A column-wise search for switching operation should be executed in the direction of most significant bit (MSB) to least significant bit (LSB). In the case of occurrence of switching, a partition should be made by a horizontal line as shown in Figure 2.

2.1.3. Step (3): Assigning default states F1 and F0

The unspecified states in FSM make the system unpredictable [2], [3], [16]. To avoid this problem default states should be defined. According to the occurrence of switching the operation of assigning default states F1 (i.e. first one) and F0 (i.e. first zero) should be performed by looking at the first and second column. It has been presented in Figure 3. As the mentioned default states are the frequently visited states, so a technique referred to in [17] can be used to protect these states and avoid soft errors.

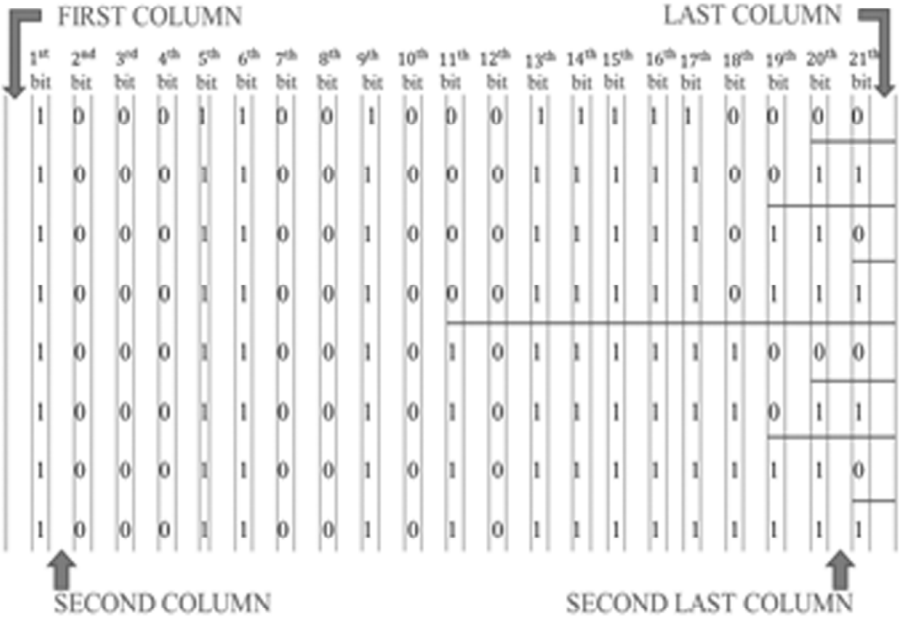

Figure 2: According to the occurrence of switching, making a column wise separation and partition of bits for the part-1 of example 1.

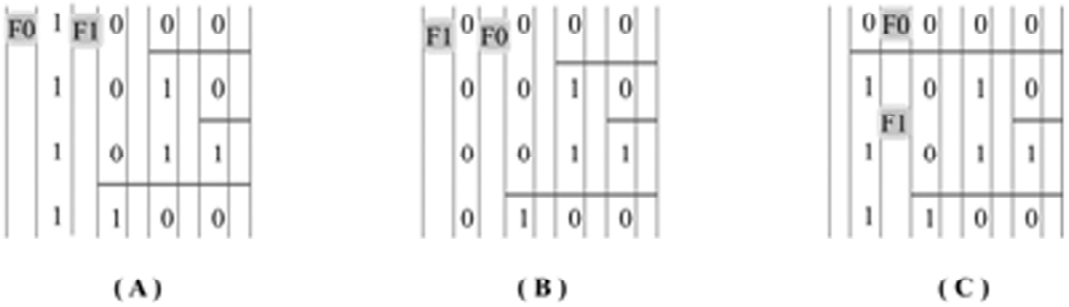

Figure 3: (A) Assigning F1 and F0, if switching does not occur and 1 is present between the first two columns. (B) Assigning F1 and F0, if switching does not occur and 0 is present between the first two columns. (C) Assigning F1 and F0, if switching occurs between the first two columns for the part-1 of example 1.

2.1.4. Step (4): Drawing equivalent state diagram

Further, an operation similar to assign names in Figure 2 should be performed to get Figure 4, which is equivalent to its state diagram. To construct a state table from Figure 4, it has been considered that at state S0 which is the initial state, if input 1 occurs, it goes to state S1, if input 0 occurs, it goes to state S0. In the case of state S1, if input 1 occurs, it goes to state S1, if input 0 occurs, it goes to state S2. Similarly, In the case of state S2, if input 1 occurs, it goes to state S1, and if input 0 occurs, it goes to state S3. It should be remembered that, if any state is not defined for input 1, then, in that case, it goes to state assigned with F1. Similarly, if any state is not defined for input 0, then, in that case, it goes to state assigned with F0. For example, states like S0, S4, S5, S8, etc. are not defined for input 0, so they go to the state S0 upon the occurrence of input 0 because S0 is assigned to F0. Similarly, states like S1, S2, S3, S6, etc. are not defined for input 1, so they go to the state S1 upon the occurrence of input 1 because S1 is assigned to F1. In the case of states present in the last column, if input 1 or 0 occurs, they go to corresponding default states.

All the desired control vectors are detected only in the states present in the second last column, at the occurrence of a particular input. Output is defined by 1 (i.e. Single output combination) if input 0 occurs at state S31 and S34, and output is determined by 1 if input 1 occurs at state S32 and S35. In the case of states S33 and S36 output is defined by 1 for input 1 or 0. For rest of the states, output is determined by 0 for both inputs i.e. 0 and 1 respectively.

2.1.5. Step (5): Applying first state reduction rule

The first state reduction [4] rule is stated as, “For the proposed detector, the states present in the first and last column of the equivalent state diagram are identical to each other and can be combined [2], [3], [8].”

2.1.6. Step (6): Applying second state reduction rule

The second state reduction rule is stated as, “To identify the identical states blocks should be created, and then the comparison of blocks to each other should be executed. The column-wise state reduction should be performed in the direction LSB to MSB.”

- Iteration (1): As all the desired control vectors are defined by a single output combination, so at this stage the following three cases are possible:

  Case (i): If more than one state will go to initial state for input 1, whereas they are not defined for input 0, then those states will be equivalent to each other [4], [8].

  Case (ii): If more than one state will go to initial state for input 0, whereas they are not defined for input 1, then those states will be equivalent to each other [4], [8].

  Case (iii): If more than one state will go to initial state for input 1 and 0, then those states will be equivalent to each other [4], [8]. It has been shown in Figure 5.

- Iteration (2): At this stage, the output is equal to 0 for all the states present in this iteration, so if more than one state goes to same next states for input 0 and 1 respectively, then they will be equivalent to each other [2]–[4].

Similarly, these iterations should be performed until the advent of states present in the second column. These state reduction process will take maximum (m-1) iterations.

A reduced equivalent state diagram is obtained at the last iteration. After finishing state reductions as mentioned earlier, a reduced state table should be formed from its reduced equivalent state diagram as shown in Table 1.

2.1.7. Step (7): Detection of m-bit long all binary control vectors by an output 1

It is the unique condition. If the detector is formed to detect m-bit long all binary vectors by an output 1, it acts as an m-bit counter, whose output is the AND gate output, which is fed by all outputs of flip-flops present in the m-bit counter. Only m states are required to detect all binary vectors of length m-bit as shown in Figure 6.

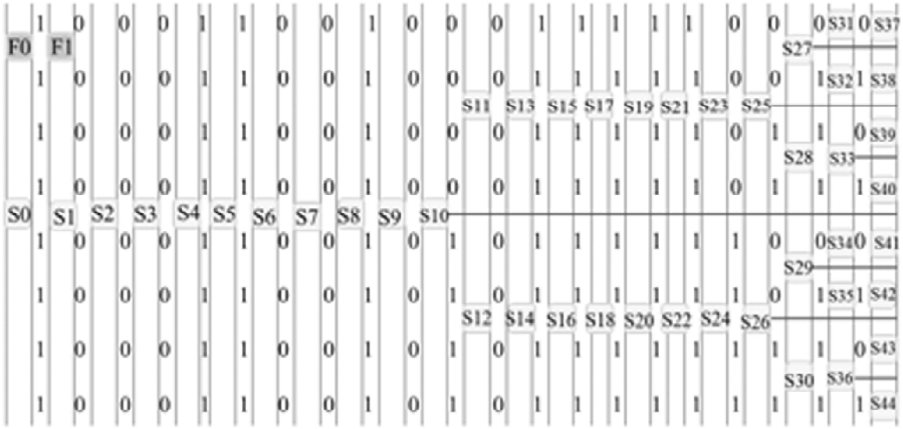

Figure 4: Equivalent state diagram for the part-1 of example 1.

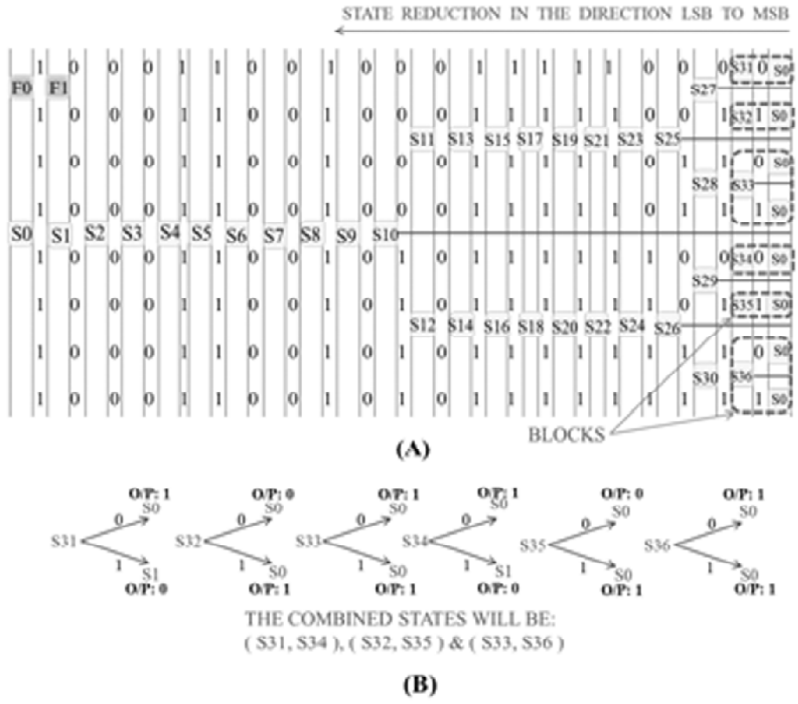

Figure 5: (A) Formation of blocks and (B) Combined states after comparing blocks to each other at iteration 1 for the part-1 of example 1.

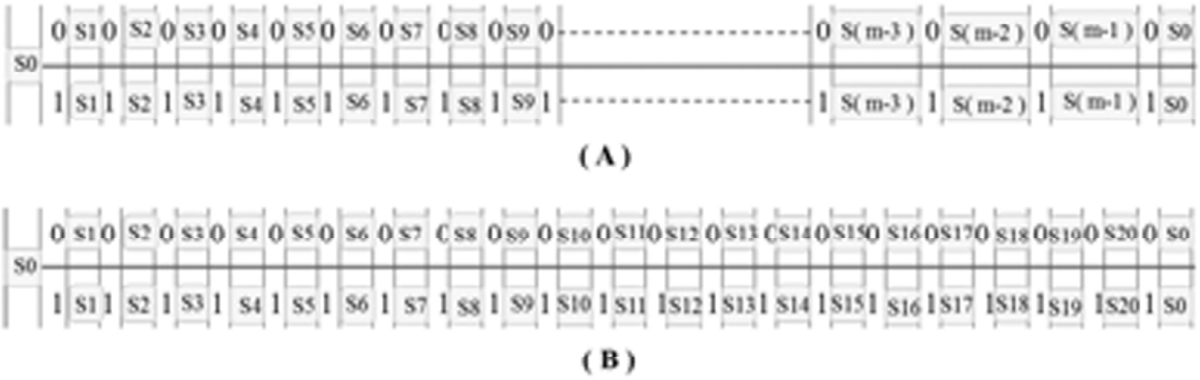

Figure 6: (A) Reduced equivalent state diagram to detect m-bit long all binary vectors by an output 1, (B) Reduced equivalent state diagram for the part-2 of example 1.

2.2. Detection of control vectors by User Defined Output Combinations

Except the second state reduction rule at iteration 1, all rules applied for detection of vectors by user defined output combinations are identical to rules used for detection of vectors by a single output combination.

Example 2. Let us consider a detector for detection of non-overlap binary control vectors of constant length 21-bit by user defined output combinations. The part-1 of the detector will produce output 1000 if binary vector 100011001000111110000 is detected. It will produce output 0100 if vector 100011001010111111000 is identified. It will produce output 0010 if any one of the following vectors is determined: 100011001000111110011, 100011001010111111011 and it will produce output 0001 if any one of the following vectors is detected: 100011001000111110110, 100011001000111110111, 100011001010111111110, 100011001010111111111. The part-2 of the detector will produce output 1 if any 21-bit vector is identified.

Table 1
Reduced State table for example (1)

PRE-SENT STATE	*NEXT STATE*		*OUTPUT z*		*PRE-SENT STATE*	*NEXT STATE*		*OUTPUT z*	
	x = 0	*x = 1*	*x = 0*	*x = 1*		*x = 0*	*x = 1*	*x = 0*	*x = 1*
S0	S0	S1	0	0	S16	S0	S18	0	0
S1	S2	S1	0	0	S17	S0	S19	0	0
S2	S3	S1	0	0	S18	S0	S20	0	0
S3	S4	S1	0	0	S19	S0	S21	0	0
S4	S0	S5	0	0	S20	S0	S22	0	0
S5	S0	S6	0	0	S21	S0	S23	0	0
S6	S7	S1	0	0	S22	S0	S24	0	0
S7	S8	S1	0	0	S23	S25	S1	0	0
S8	S0	S9	0	0	S24	S0	S25	0	0
S9	S10	S1	0	0	S25	S27	S28	0	0
S10	S11	S12	0	0	S27	S31	S32	0	0
S11	S13	S1	0	0	S28	S0	S33	0	0
S12	S14	S1	0	0	S31	S0	S1	1	0
S13	S0	S15	0	0	S32	S0	S0	0	1
S14	S0	S16	0	0	S33	S0	S0	1	1
S15	S0	S17	0	0					

The second state reduction rule at iteration 1 for detection of control vectors by user defined output combinations can be specified as, Two states S_i and S_j are said to be equivalent if they go to same next states and produce same output combinations for input 0 and 1 respectively [2], [3], [5] as shown in Figure 7.

3. ESTIMATION OF MEMORY ELEMENTS REQUIRED FOR IMPLEMENTATION OF THE PROPOSED SEQUENCE DETECTOR

A formula is presented for calculation of an approximate number of memory elements or flip-flops required for implementation of the proposed detector. The presented formula gives the exact number of memory elements needed for a detector with a single output combination without applying second state reduction rules. The number of reduced states after applying second state reduction rule and number of additional states required for implementation of the detector by user defined output combinations are not reasonably high. So the proposed formula gives an approximate measure of required memory elements because it has been described in logarithmic scale. The proposed formula is applicable only after applying the first two steps of sequence detection. The proposed formula for estimation of required memory elements is given by Equation (1).

$$\begin{aligned} &\textit{Number of required memory elements} = \\ &\Big\lceil \log_2 \big\{ m[\, m + (m - \textit{first switching between seq.1 \& seq.2}) + \\ &\quad (m - \textit{first switching between seq.2 \& seq.3}) + \\ &\quad (m - \textit{first switching between seq.3 \& seq.4}) \\ &\quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + (m - \textit{first switching between seq.}(n-1) \textit{ \& seq.}n)] \big\} \Big\rceil \end{aligned} \tag{1}$$

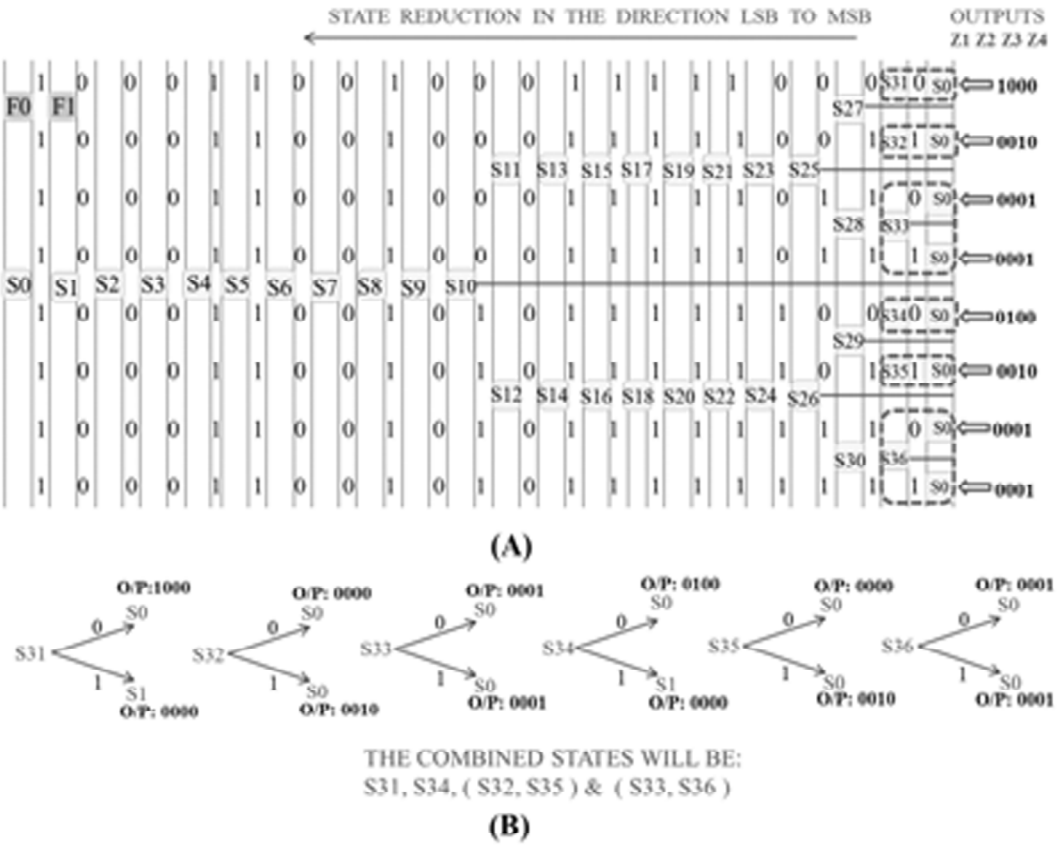

Figure 7: (A) Formation of blocks and (B) Combined states after comparing blocks to each other at iteration 1 for the part-1 of example 2.

4. NEW APPROACH FOR DEVICE PORTABILITY

A new approach is illustrated for device portability. Let us consider a system with a single or multiple general purpose processor units (GPUs) and limited input/output ports. GPU generates different constant length and non-overlap binary control vectors corresponding to particular processes as safe control commands [15]. It has been taken into account that output AC/DC characteristics of a particular intermediate device must match with the input AC/DC characteristics of the corresponding output device. Hence, along with the modification of GPU programming, the proposed detector can be studied to add new devices or to modify the previously defined functionalities of existing output device or to perform both on a single port. Device portability comes with a trade-off of the speed of operation.

Example 3: Let us consider a robot that consists of a GPU, which generates 8-bit length control signals from port-1 to operate the output units as shown in Figure 8. Extra devices can be added to the robot by introducing a new intermediate device (i.e. proposed detector) and output features of the robot can be modified by changing the configuration of the intermediate device along with the amendment in GPU programming. Operational description [10] for example 3 has been presented in Table 2.

Table 2
Operational description for the example (3)

Device Name	Robot part name	Job (or Process) to beperformed	Process Defined by Sequence	Output Combinations
Inter-mediate Device 1	Arm Control	Rotate Clockwise	00000001	z1_1 z2_1= 10
		Rotate Anti-clockwise	00000010	z1_1 z2_1= 01
Inter-mediate Device 2	Direction Control	Move Forward	00000011	z1_2 z2_2 z3_2 z4_2 = 1000
		Move Backward	00000100	z1_2 z2_2 z3_2 z4_2 = 0100
		Move Left	00000101	z1_2 z2_2 z3_2 z4_2 = 0010
		Move Right	00000110	z1_2 z2_2 z3_2 z4_2 = 0001
Inter-mediate Device 3	Speed Control	Speed Level 1	00000111	z1_3 z2_3 z3_3 z4_3 z5_3 = 10000
		Speed Level 2	00001000	z1_3 z2_3 z3_3 z4_3 z5_3 = 01000
		Speed Level 3	00001001	z1_3 z2_3 z3_3 z4_3 z5_3 = 00100
		Speed Level 4	00001010	z1_3 z2_3 z3_3 z4_3 z5_3 = 00010
		Speed Level 5	00001011	z1_3 z2_3 z3_3 z4_3 z5_3 = 00001

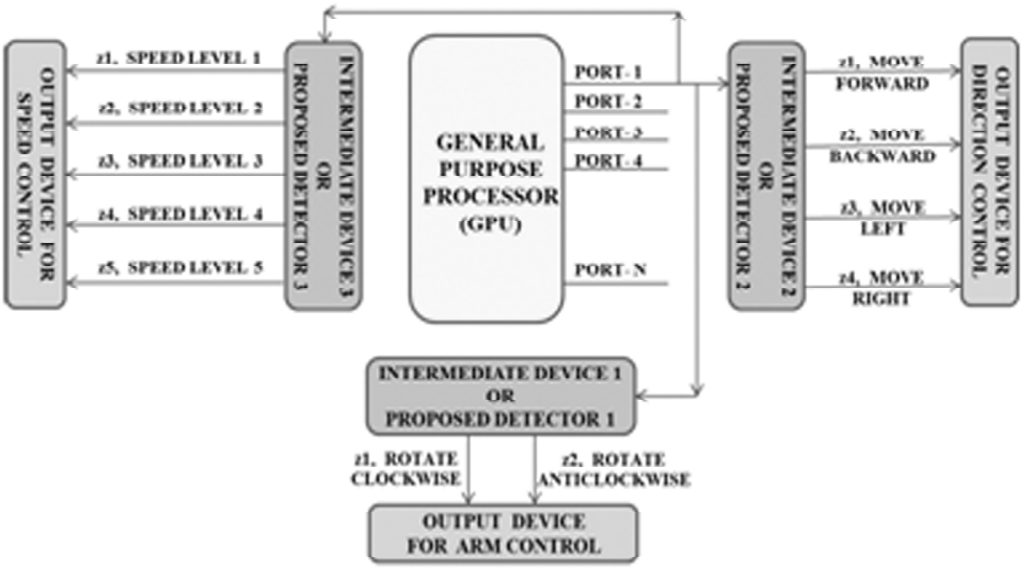

Figure 8: System model for example 3.

5. RESULTS AND DISCUSSIONS

The variation of the number of required states for FSM implementation of part-1 w.r.t. possible combinations of adjacent binary vectors are shown in Figure 9. Comparison results have been presented in Figure 10. In Figure 9 & Figure 10, an up-counting scheme have been used for generation of all possible combinations of sequences after applying first two steps of sequence detection, and its decimal equivalent have been used for representation as shown below:

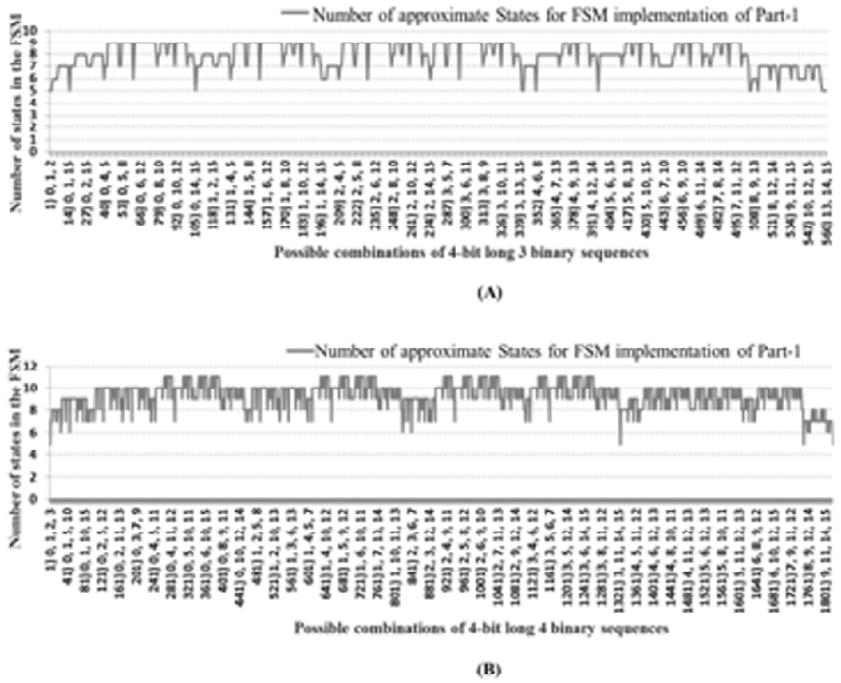

Figure 9: (A) For 4-bit long 3 sequences and (B) For 4-bit long 4 sequences, variation in state requirement for FSM implementation of part-1 w.r.t. all possible combinations of adjacent sequences.

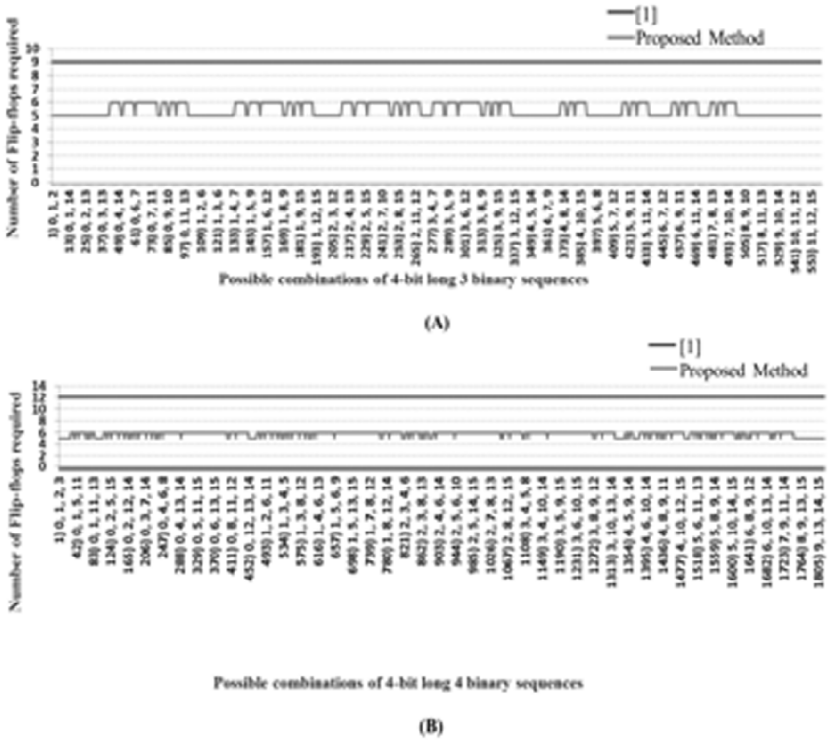

Figure 10: (A) For 4-bit long 3 sequences and (B) For 4-bit long 4 sequences, comparison of memory element requirement between proposed method and the existing method

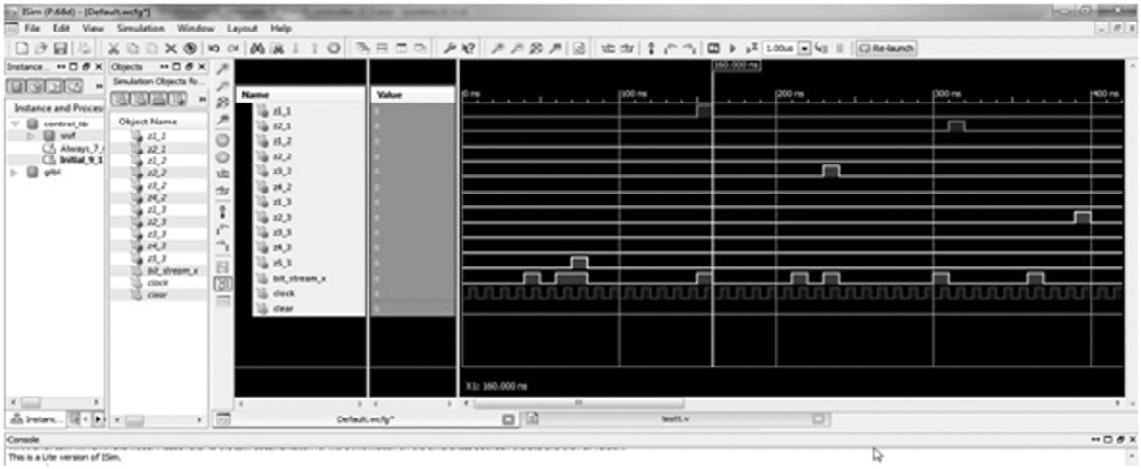

Figure 11: Simulation results for example 3.

(0, 1, 2), (0, 1, 3), (0, 1, 4), ..., (3, 4, 15), (3, 5, 6), (3, 5, 7), ..., (12, 14, 15), (13, 14, 15).

The program written for examples as mentioned earlier have been successfully simulated using Xilinx ISE Design Suite. Simulation results have been presented for example 3 in Figure 11 (Note: all variables contain their usual meaning). Hardware implementations have been performed using Spartan 3E Starter Board; which validates the simulation results. Table 3 presents different parameters for it.

6. CONCLUSION

The proposed technique provides an optimal hardware compared with existing method when the number of desired control vectors and correlation between them is high. Therefore, it can be applied to design low power devices. The states combined in a particular iteration is proportional to the states combined in its previous iteration while state reduction. Therefore, the number of states required to implement the FSM for proposed detector by user defined output combinations is always greater than the number of states needed

Table 3
Device Utilization Summary for example 3

	Device Utilization Summary		
Logic Utilization	*Used*	*Available*	*Utilization*
Number of Slice Flip Flops	47	9,312	1%
Number of 4 input LUTs	56	9,312	1%
Number of occupied Slices	36	4,656	1%
Number of Slices containing only related logic	36	36	100%
Number of Slices containing unrelated logic	0	36	0%
Total Number of 4 input LUTs	56	9,312	1%
Number used as logic	53		
Number used as Shift registers	3		
Number of bonded IOBs	14	232	6%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	2.78		
Maximum Operational Frequency	249.128 MHz		
Path Delay	6.959 ns		
Memory Usage	210.8 MB		
Static Power Consumption	0.081 W		

to implement it by a single output combination. It can be used for process isolation in any reconfigurable processing fabric architectures (RPFs) [18]. It can be used for designing more efficient sampled correlator [19]. To avoid race conditions in asynchronous digital systems a hybrid design structure can be made which combines synchronous (i.e. proposed technique) and asynchronous digital design [8]. In the proposed scheme for device portability, the speed of operation of the system is proportional to the clock frequency of GPU. Also, it consists of only a single input port and user defined output ports, therefore, it may be used for multiplexing operations.

REFERENCES

[1] Marvin Perlman, "Binary Sequence Detector," US3493929 A, 1970.

[2] W. T. Shiue, "Novel state minimization and state assignment in finite state machine design for low-power portable devices," *Integr. VLSI J.*, vol. 38, no. 4, pp. 549–570, 2005.

[3] W. Shiue, "Power / area / delay aware FSM synthesis and optimization," vol. 36, pp. 147–162, 2005.

[4] I. Ahmad, F. M. Ali, and A. S. Das, "Synthesis of finite state machines for improved state verification," *Comput. Electr. Eng.*, vol. 32, no. 5, pp. 349–363, 2006.

[5] S. Brown and Z. Vranesic, *Fundamentals of digital logic with VHDL design*. Third Edi., McGraw Hill Higher Education, New York, 2005.

[6] R. Czerwinski and D. Kania, "Area and speed oriented synthesis of FSMs for PAL-based CPLDs," *Microprocess. Microsyst.*, vol. 36, no. 1, pp. 45–61, 2012.

[7] P. K. Biswal and S. Biswas, "A Binary Decision Diagram based on-line testing of digital VLSI circuits for feedback bridging faults," *Microelectronics J.*, vol. 46, no. 7, pp. 598–616, 2015.

[8] C. Cao and B. Oelmann, "Low-power state encoding for partitioned FSMs with mixed synchronous/asynchronous state memory," *Integr. VLSI J.*, vol. 41, no. 1, pp. 123–134, 2008.

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

[10] D. Patel, J. Bhatt, and S. Trivedi, "Programmable logic controller performance enhancement by field programmable gate array based design," *ISA Trans.*, vol. 54, pp. 156–168, 2015.

[11] M. Rawski, H. Selvaraj, and T. £uba, "An application of functional decomposition in ROM-based FSM implementation in FPGA devices," *J. Syst. Archit.*, vol. 51, no. 6–7, pp. 424–434, 2005.

[12] M. Kolopieñczyk, A. Barkalov, L. Titarenko, and M. Wegrzyn, "Hardware reduction for EMB-based Mealy FSM," *IFAC-PapersOnLine*, vol. 48, no. 10, pp. 202–207, 2015.

[13] A. Barkalov, L. Titarenko, K. Mielcarek, and M. Wegrzyn, "Design of EMB-based Mealy FSMs with transformation of output functions," *IFAC-PapersOnLine*, vol. 48, no. 10, pp. 197–201, 2015.

[14] V. Sklyarov and ', "Reconfigurable models of finite state machines and their implementation," *FPGAs'. J. Syst. Archit.*, vol. 47, pp. 1043–1064, 2002.

[15] S. Hajjar, E. Dumitrescu, L. Pietrac, and E. Niel, "A design method for synthesizing control-command systems out of reusable components," *Adv. Astronaut. Sci.*, vol. 12, pp. 60–65, 2014.

[16] M. A. Perkowski, "Symbolic two-dimensional minimization of strongly unspecied finite state machines," *J. Syst. Archit.*, vol. 47, pp. 15-28, 2001.

[17] A. H. El-Maleh and A. S. Al-Qahtani, "A finite state machine based fault tolerance technique for sequential circuits," *Microelectron. Reliab.*, vol. 54, no. 3, pp. 654–661, 2014.

[18] S. Hauck and Andre DeHon (Ed.), *Reconfigurable computing: the theory and practice of FPGA-based computation*. Morgan Kaufmann Publishers, Burlington, MA, 2007.

[19] B. Sklar, *Digital Communications: Fundamentals and Applications*, Second Edi. Upper Saddle River, New Jersey, 2009.

[20] S. F. Beldianu and S. G. Ziavras, "Multicore-Based Vector Coprocessor Sharing for Performance and Energy Gains," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, pp. 1-25, September 2013.