Implementing State Machines in LCA Devices

XAPP 027.001

Application Note BY PETER ALFKE AND BERNIE NEW

Summary

This Application Note discusses various approaches that are available for implementing state machines in LCA devices. In particular, the one-hot-encoding scheme for medium-sized state machines is discussed.

LCA Family

XC3000/XC3100/XC3100A

Demonstrates

State Machine Design
One-hot Encoding

Introduction

State-machine methodology defines the contents of every flip-flop in a design under every circumstance that might arise. It also defines all the possible transitions that can cause the design to go from one of these states to another. In its simplest form, this is just a rigorous way of designing synchronous logic, like 4-bit counters. For more complex designs, the state-machine approach gives the designer a tool to analyze all possible operating conditions, and so avoid overlooked hang-up states or undesired transitions. LCA devices with their abundance of flip-flops lend themselves well to state-machine designs.

Using the 5-input function generator of the XC3000 family devices as a 32-bit ROM, a state machine with up to 32 states with no conditional jumps uses only five CLBs. Five registered CLB outputs drive the five function generator inputs of five CLBs in parallel. This implements a fully programmable sequencer such as a synchronous counter.

For a smaller number of states, some inputs can be used as conditional jump inputs. Encoding these condition codes, however, may require an additional level of logic which reduces the maximum clock rate.

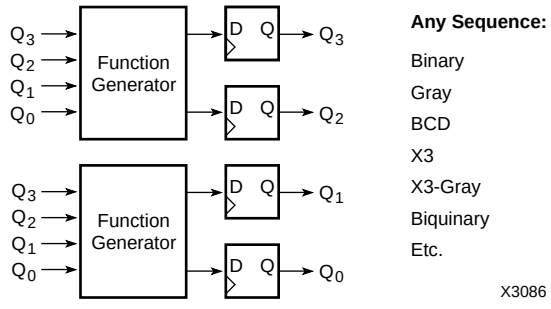

Figure 1. Synchronous 4-Bit Counter in 2 CLBs

Synchronous Counters

Using only two CLBs, it is possible to construct fully synchronous 4-bit counters with arbitrary count sequences, Figure 1. The CLB Clock Enable inputs even provide count-enable control. The count length, count direction, and even the code sequence is determined by the configuration. The number of possible count sequences is factorial 15, i.e., more than 10^{12}. All four outputs are available, and while the counter cannot be preset to an arbitrary value, it can be cleared by an asynchronous input.

Table 1 shows four common count sequences. Of particular interest is the Gray code, which offers glitch-less decoding, since only one bit changes on any transition. A Gray-code counter can also be reliably read asynchronously. In contrast, if a binary counter is read during its transition between 7 and 8, for example, any code might be detected.

Decimal	Binary	Gray	X3 Binary	X3 Gray
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0011	0101	0111
3	0011	0010	0110	0101
4	0100	0110	0111	0100
5	0101	0111	1000	1100
6	0110	0101	1001	1101
7	0111	0100	1010	1111
8	1000	1100	1011	1110
9	1001	1101	1100	1010
10	1010	1111		
11	1011	1110		
12	1100	1010		
13	1101	1011		
14	1110	1001		
15	1111	1000		

Table 1. Four Common Binary Count Sequences

Four-bit counters constructed as described above can easily be concatenated into longer, four-bits-at-a-time ripple-carry counters. For each 4-bit digit, a third CLB is used to detect an arbitrary terminal count value, and AND this with the incoming Count Enable to provide the Count Enable to the next digit.

Waveform Generator

Arbitrary binary waveforms of any length up to 32 clock periods can be generated using only three XC3000 series CLBs, Figure 2. The waveform generation is fully synchronous, and may be paused at any time, using the CLB Clock Enable. It may also be restarted, using the asynchronous clear.

Five flip-flops, Q_{0-4}, form a linear feedback shift-register counter. The 5-input combinatorial function generator, F_0, determines both the modulus and the count sequence; there are no illegal or hang-up states. The function generator, F_1, operates as a ROM, and can be programmed to provide any conceivable decode of the counter. Flip-flop, Q_5, synchronizes and de-glitches the decoder output.

The following examples demonstrate the arbitrary nature of the waveforms that can be generated.

Example 1. + 28 counter with its output High at times T2, T3, T10, T22 through T27

Example 2. +19 counter with its output Low at times T9, T12, T15, T18.

Simple State Machines

The simple state machine shown in Figure 3 uses only 10 CLBs, and has up to 16 states. Each of eight outputs decode/encode any combination of states. The state machine is based on a 5-CLB next-state look-up table.

Each state corresponds to two look-up table locations that store two arbitrarily defined next states. From any state, the C input controls a two-way branching by selecting which of the two possible next states is asserted. For hold loops, one of the next states should be the current state; and to avoid branching, both destination states should be made equal.

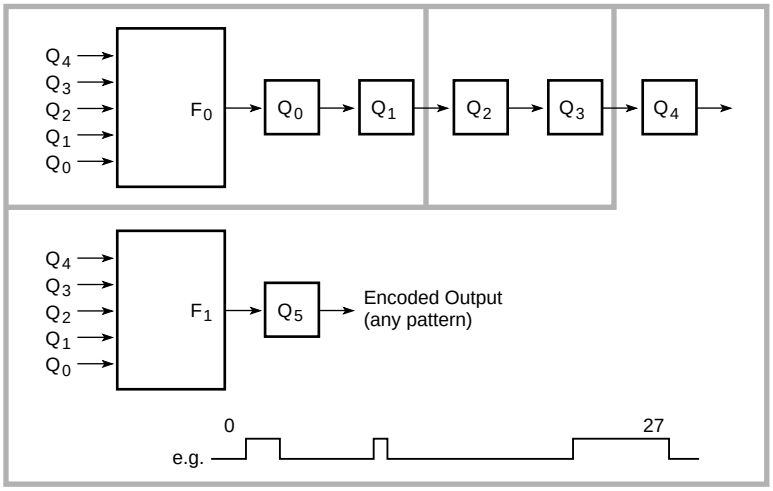

Figure 2. Synchronous 5-Bit Waveform Generator in 3 CLBs

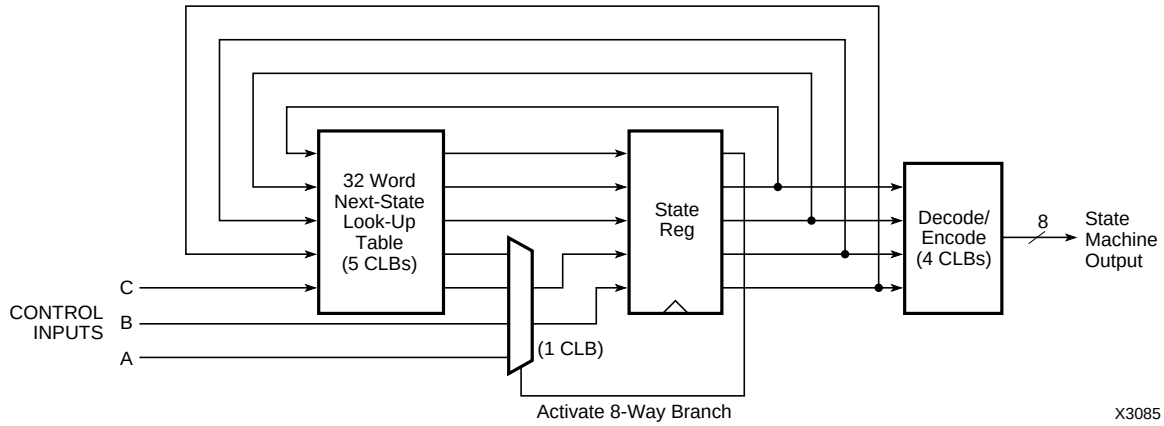

Figure 3. Simple Sate Machine

The state machine can also perform 8-way branches from any state so programmed. The branch destinations must all fall in two quadrants (0..3, 4...7, 8...11 or 12...15). The choice of the two quadrants is arbitrarily programmed into the look-up table; C selects between the two quadrants, and A and B select the state within the quadrant.

Activation of the 8-way branch mechanism is controlled by a fifth state bit that is set during the transition into the state. This bit controls a multiplexer that replaces the two LSB of the destination state with the control inputs A and B. Note that as the fifth bit is independent of A and B, it must be set, or not, on a per quadrant basis during an 8-way branch.

Examples:

- From state 3, if C = High, go to 5, else go to 8
- From state 7, if C = High, go to 3, else stay in 7
- From state 9, unconditionally go to 2
- From state 6, execute the truth table below

Truth Table

A	B	C = Low	C = High
0	0	12	0
1	0	13	1
0	1	14	2
1	1	15	3

One-Hot Encoded State Machines

The state machines described have encoded state bits. For an N-state state machine, fewer than N flip-flops are used (but $\geq \log_2 N$), and a unique combination of these flip-flops is set in each state; each flip-flop is set in several states. While this minimizes the number of flip-flops, it increases the complexity of the logic controlling each flip-flop.

In LCA devices, flip-flops are plentiful, and there is no need to conserve them. Consequently, for medium-sized state machines, it is better to use a One-Hot encoding scheme (OHE). OHE increases the number of flip-flops required, but reduces the logic complexity associated with each of them, thereby boosting performance.

In an OHE state machine, one flip-flop is assigned to each state. It is set during that state, and only during that state. The state machine is implemented as a shift-register-like structure, where a single One is passed from flip-flop to flip-flop, sometimes holding in the same flip-flop, skipping bits of the shift register or moving to a parallel shift register, Figure 4a and b.

The control logic associated with each state bit involves ORing the transitions into the state, including any hold loop. Each of these transitions will involve a previous state, which, by design, is represented by a single bit. This bit may, or may not, be ANDed with some decode of the control bits inputs.

It is the localization of the control logic that leads to the performance increase. For each state bit, the control logic only involves the limited number of state bits from which there are transitions and the conditions that control those transitions. This permits shallow logic structures between flip-flops, often only requiring the function generator associated with the state-bit flip-flop. In addition, no state decoding is necessary, and state encoding can only require the ORing of state bits.

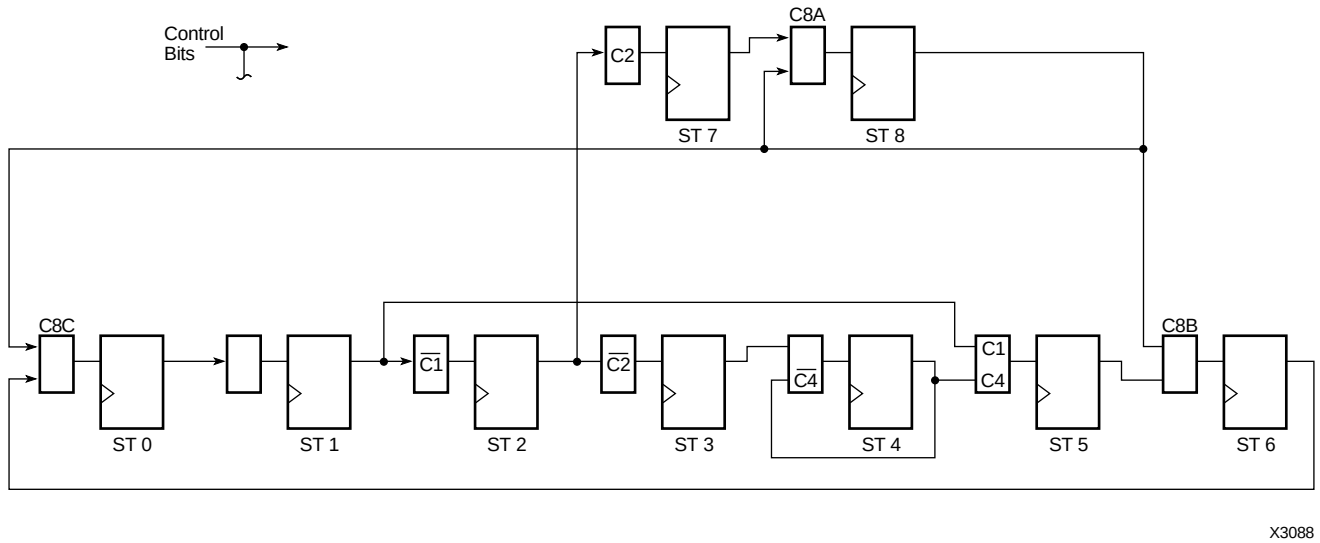

Figure 4a. Prototype OHE State Machine

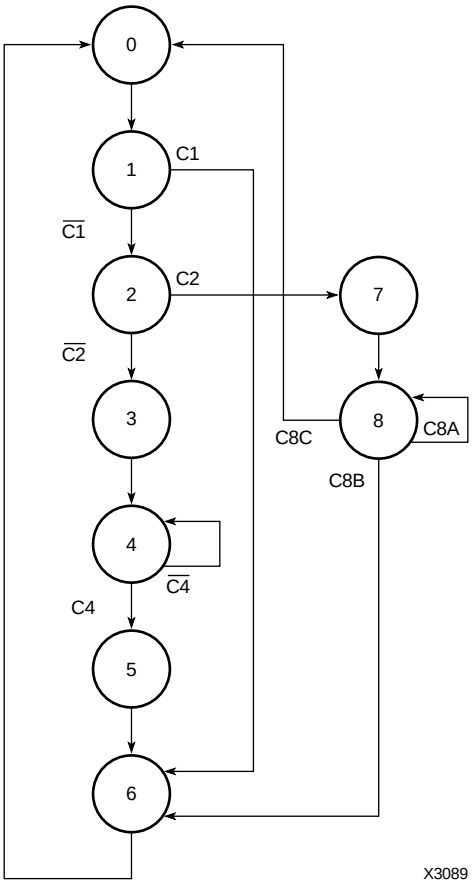

Figure 4b. State Diagram for Prototype OHE State Machine

Complex State Machines

Small- and medium-sized state machines can easily be implemented within an LCA device, as shown above. For large, complex state machines, however, it is better to use the LCA device to implement a simple microsequencer, and store the control program externally, Figure 5.

For fastest operation, a high-speed SRAM should be used for the control program. This may be loaded from a microprocessor, or shadowed by an EPROM. For slower operation requiring non-volatility, an EPROM can be used directly. When an EPROM is used, the number of components can be reduced by storing both the LCA configuration data and the state-machine control program in the same device.

If an XC3020 is configured in the Master Parallel mode and it reads its configuration data out of a 256K (32K x 8) EPROM, it only requires 6% of the addresses, from the top location 7FFF (32K) through 77FF (about 30K). The remaining 94% of the EPROM can be used as a next-state look-up table with a capacity of 240 states.

Eight state bits are read out of the EPROM and registered in the LCA device which can perform any required decoding

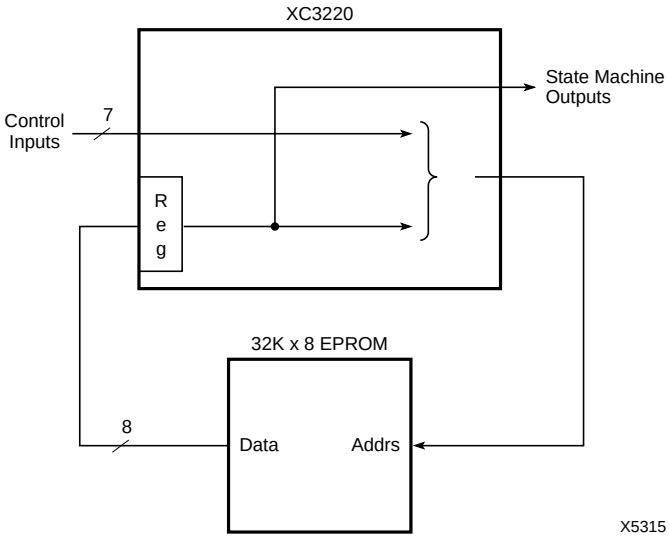

Figure 5. Rudimentary Complex State Machine

or encoding of the state-machine outputs. The registered state bits also form part of the new EPROM address, defining a block of 128 possible next states. The 7-bit condition code completes the EPROM address and selects which of 128 next states is actually asserted.

Each transition is, in effect, a 128-way branch. However, the branching complexity will normally be reduced by assigning identical values to many of the 128 possible next states.

Since the top 16 address locations are used for configuration data, the state codes, which form the 8 MSBs of the EPROM address, are limited to 240 different values, 0...239. The control inputs provide the seven LSBs of the EPROM address. If the control inputs are asynchronous, they must be registered for reliable operation.

This rudimentary state machine can thus have 240 different states, and can jump from any state to any one of 128 arbitrarily defined next states, according to a 7-bit condition code. In its simplest form, this basic design consumes no CLB resources in the LCA, just IOB flip-flops for the state register. Even so, it permits a number of states and a multiway branch complexity far in excess of any normal need.

The user has all the logic resources of the LCA available to add features like the following.

- State decoding/encoding
- Stack registers
- Loop counters
- More sophisticated branch logic, etc.

This design is straightforward, inexpensive, compact and extremely flexible. Its speed is limited primarily by the control store access time; faster access times can be obtained using SRAMs in place of EPROMs.