Binary Sequence Detector

Authors: Ing. Alberto I. Leibovich & Ing. Pablo E. Leibovich
Date: November 14, 2016

Introduction

In baseband digital communication channels, binary serial transmission is the most prevalent way to share information between transmitters and receivers. These serial channels are used in networks where computers, embedded systems and even IoT (Internet of Things) devices can be connected via a number of standard protocol implementations, such as TCP, UDP, RS232, etc.

In channels where binary data is sent one bit at a time at high data rates, the binary transmission is referred to as a stream and the carried information is called a packet. The challenge of this type of transmission is defining the start and end of the transmission. In general, flags are used to allow the receiver to identify the start and end of the packet.

To achieve this objective, bit sequences are defined to identify the beginning and ending of a message and they are used to set or clear the flags. Binary sequence detectors are used to detect these sequences at the receiving end.

This application note shows how to implement a design using the GreenPAK based on a state machine. In this example, the pattern "101" gets detected from a binary stream.

State Machine Design

In digital design, there are Combinational circuits and Sequential circuits. The former operate using only logic functions of the inputs, without any dependency on previous states. Whereas, in the latter category of circuits, the output at any stage is dependent on the previous states, which means that certain memory elements are involved in the circuit.

That's the case in binary sequence detectors as well, where previous bits have to be used to detect the desired sequence.

In Sequential circuit design, it is important to define whether the output of the system depends only on the present state, or if it also depends on the current input. These two structural possibilities are known respectively as Moore or Mealy state machines. In Moore machines, the output depends only on the present state and doesn't care about the current input. In Mealy machines, the output depends both on the present state and the current input. Moore machines are safer to use since outputs do not change asynchronously to a clock. This application note uses Moore machines.

In this example, the pattern "101" is detected in a binary stream (X is the input). When the sequence has not yet been detected, the output of the system will stay at a low level. When the sequence gets detected, the output turns to high until a 0 is found in the stream.

In figure 1, the Moore state diagram is shown.

State 0 (S_0) is the first state. Here, the system waits with 0 as output until the first 1 of the sequence is detected. In that case, it goes to state 1 (S_1), where it stays until a 0 is received. This is because the system can receive a stream like "11111101". When a 0 is detected, we reach state 2 (S_2). This is the final decision state. If a '0' is detected here, it means that the sequence was "100" (which wasn't the desired pattern) so the machine gets reset to state 0. If we detect a '1' here, it means that the desired pattern was received which sets the machine to state 3 (S_3), with a high level (1) at the output. The system waits in state 3 until a '0' is received.

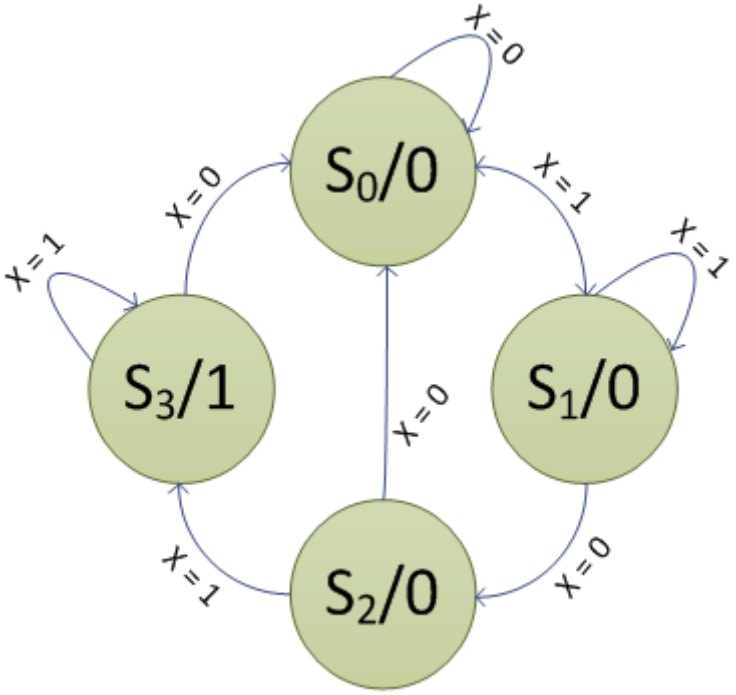


Figure 1. State Diagram

Logic Design

From the State Diagram shown in the previous section the following transition table can be obtained (See Table 1).

Actual State	Next State		Output
	X = 0	X = 1	
S_0	S_0	S_1	0
S_1	S_2	S_1	0
S_2	S_0	S_3	0
S_3	S_0	S_3	1

Table 1. Transition Table

It can be concluded that, because of the implementation comprising 4 states, the design will require 2 flip-flops. All of GreenPAK ICs have D-type flip-flops, so if the selected GreenPak doesn't have the Asynchronous State Machine (ASM) Module, then the implementation must generate the functions for the flip-flops inputs.

The logic functions depend both on the stream input and on the previous states, hence they are functions of X and Q1/Q0 (The flip-flop outputs).

For the first flip-flop (Q0 output), the truth table of the logic function must look like the one shown in Table 2.

Q1	Q0	X	D0
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 2. D0 Truth Table

For the second flip-flop (Q1 output), the truth table of the logic function is shown in Table 3.

Q1	Q0	X	D1
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 3. D1 Truth Table

Finally, for the system output (Z), the truth table of the logic function is shown in Table 4.

Q1	Q0	Z
0	0	0
0	1	0
1	0	0
1	1	1

Table 4. Z Truth Table

Implementation

We can implement GreenPAK desgin in various ways depending on the resources available in the particular GreenPAK version. We will explore implementations using GreenPAK Flip-flops, as well as the built-in ASM (Asynchronous State Machine) available in some GreenPAK's.

Flip – Flop based Implementation

In this case, 2 flip-flops and 3 lookup tables are used. Figure 2 shows the the block diagram.

In the figure, it can be seen that the input (X) is mapped to PIN 2, the clock input is mapped to PIN 3 (which is also connected to the clock input of the rising-edge-triggered flip-flops) and the output is connected to PIN 8. PIN 2 and PIN 3 are configured as Digital – In without a Schmitt trigger and without a resistor. PIN 8 is used as an output, with its Output Enable tied to VDD.

The logic function of the input of DFF0 is implemented as a LUT2; its configuration is shown in figure 3.

The logic function corresponding to the input of DFF1 is impelemented as a LUT3, it is configured as shown in figure 4.

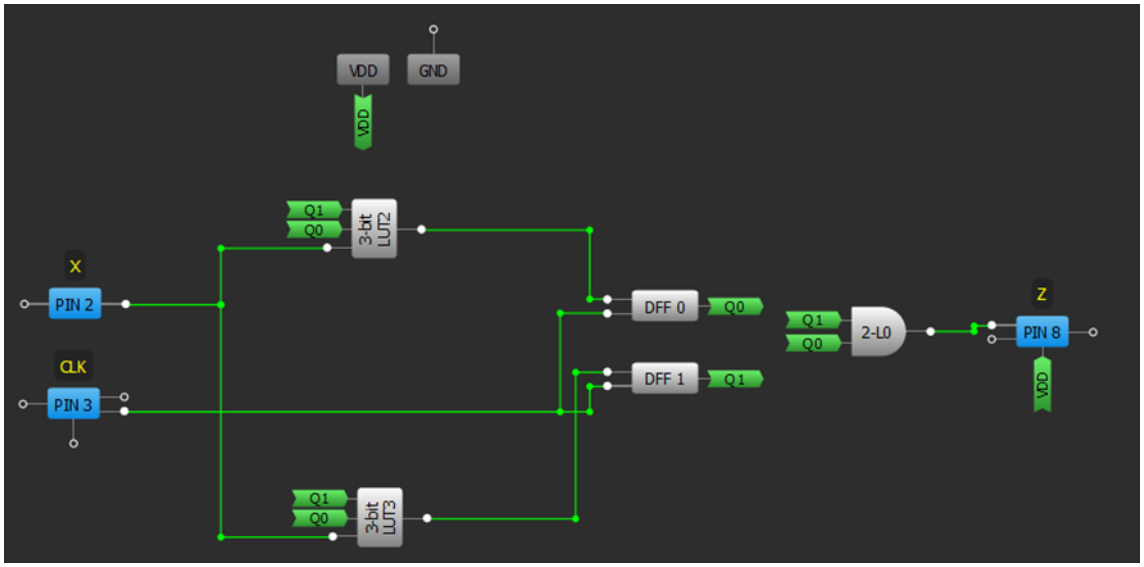


Figure 2. Flip-Flop based block diagram

3-bit LUT2

IN3	IN2	IN1	IN0	OUT
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Standard gates: Defined by user; Regular shape; All to 0; All to 1; Invert

Figure 3. LUT2 Configuration

3-bit LUT3

IN3	IN2	IN1	IN0	OUT
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Standard gates: Defined by user; Regular shape; All to 0; All to 1; Invert

Figure 4. LUT3 Configuration

Finally, the output logic function is implemented using LUT0, because we only need 2 bits in our input. The configuration can be seen in figure 5.

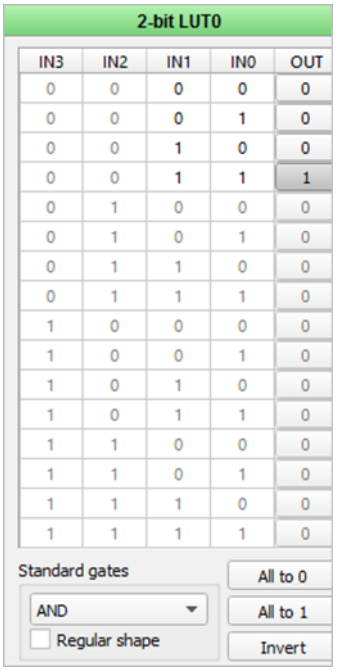


Figure 5. LUT0 Configuration

ASM based implementation

This design is based on the ASM (Asynchronous State Machine) Module available in GreenPak5. Because of the synchronous nature of the implemented system, the asynchronous nature of this module must be considered. The block diagram is shown in figure 6.

In this figure, it can be seen that the input (X) is mapped to PIN 4, the clock input is mapped to PIN 3 and the output is connected to PIN 10. PIN 3 and PIN 4 are configured as Digital – In with Schmitt trigger and without a resistor.

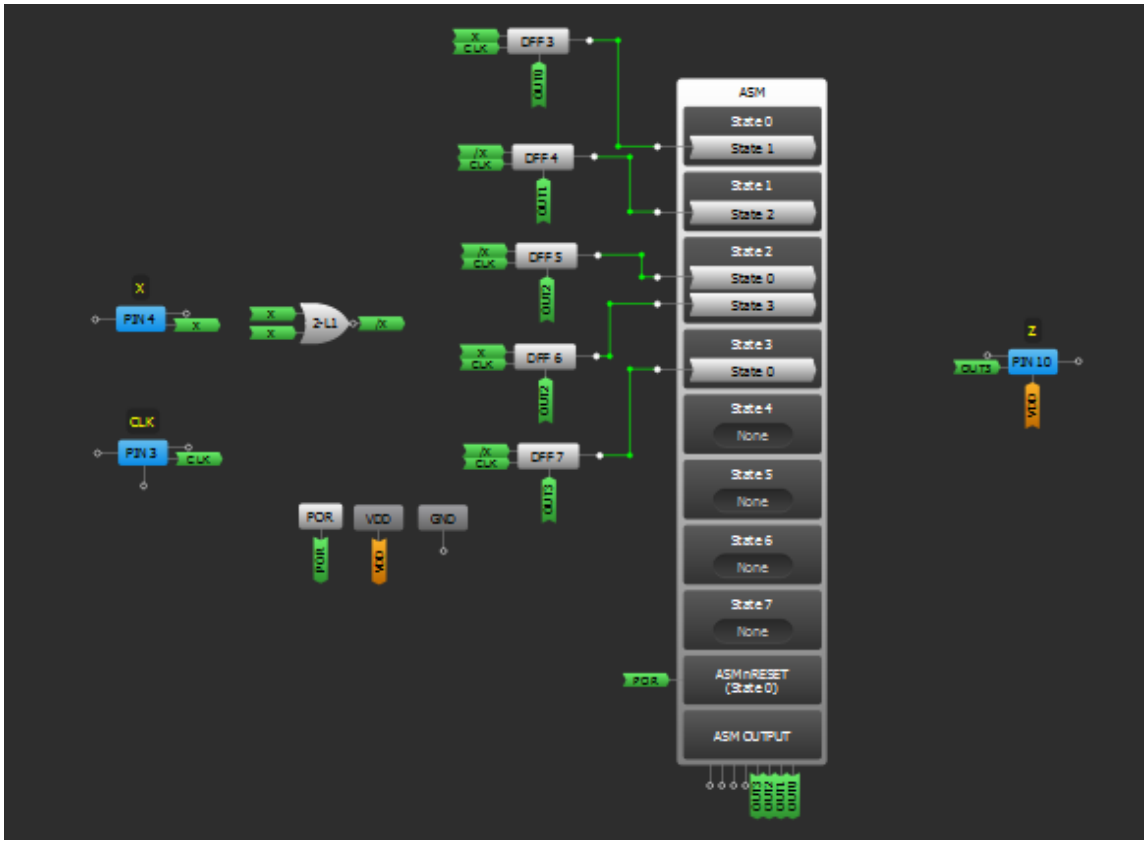


Figure 6. ASM based block diagram

PIN 10 is used as an output, with its Output Enable tied to VDD.

It's important to explain the presence of the D-type flip-flops. Because of the level-sensitive and active-high transition inputs of the ASM, flip-flops with nReset input are used to obtain a high level only when the desired transition has to be done.

Each flip-flop is reset when the corresponding state is not active (hence the output is low). When the corresponding state is active, the ASM output turns to high and the flip-flop becomes active.

A high level should be obtained at the output of the FF only when there is a rising-edge on the clock and the corresponding state of the input. To achieve this, an inversion of the signal is done with 2-bit LUT1. This helps in obtaining a high level for the transitions where the input will be low. With this scheme, the ASM is able to perform with synchronous behavior.

Based on the state machine shown in figure 1, the ASM is configured using the ASM Editor. The outputs of the states are high only when the machine is on the corresponding state. This can be seen in figure 7.

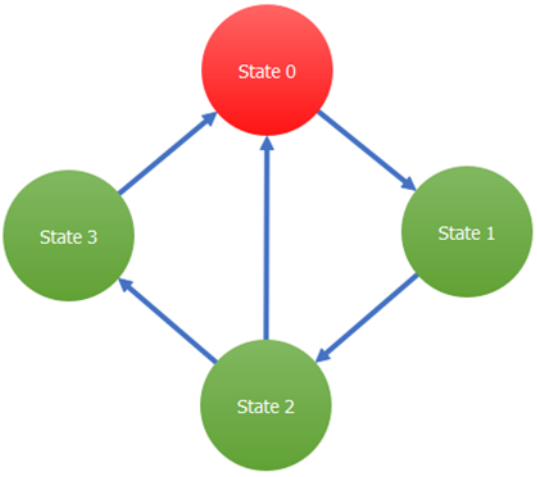
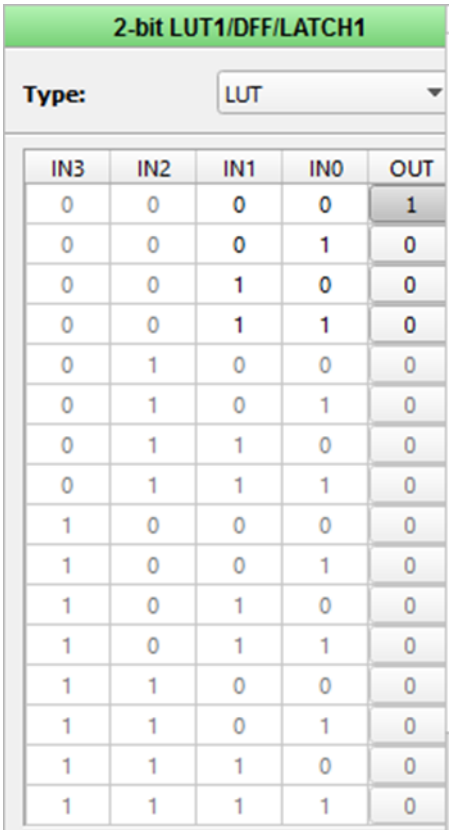


Figure 7. ASM Configuration

In figures 8 and 9, the 2-bit LUT1 and flip-flop configurations are shown.



IN3	IN2	IN1	IN0	OUT
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Figure 8. 2-bit LUT1 Configuration

DFF3 output should be 1 only when a rising edge is detected, X is high and State 0 is active; hence the transition from State 0 to State 1 is done only when these conditions are met.

DFF4 output is high when the corresponding edge is detected, X is 0 (the inverted signal is high) and State 1 is active. This output handles the transition from State 1 to State 2.

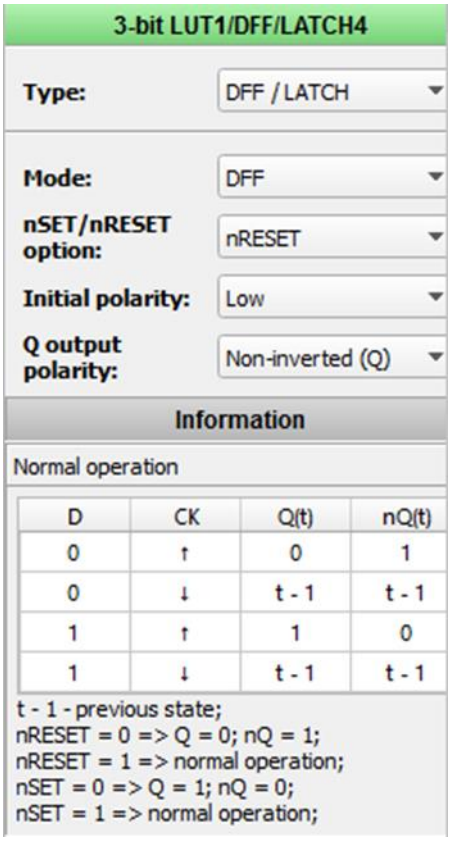


Figure 9. DFF Configuration

DFF5 controls the transition from State 2 to State 0, so it's configured to set its output high when a rising edge of clock is detected with a 0 in X and when state 2 active. DFF6 is the complement of DFFX. It controls the transition from State 2 to State 3, so its output is high when a rising edge is detected with a 1 in X and when state 2 active. Finally, DFF7 handles the transition from State 3 to State 0. It's high only when a rising edge is detected and when X is 0 with State 3 being active.

The output of this design is connected directly to the output of State 3 in the ASM because it must be high only when the state machine is on State 3.

Test and Conclusion

Both designs presented in this application note were tested with a logic analyzer while capturing the inputs and the output.

Figures 10 and 11 show the signals for the first implementation. In the figures, channel 1 is the clock, channel 2 is the input and channel 3 is the output

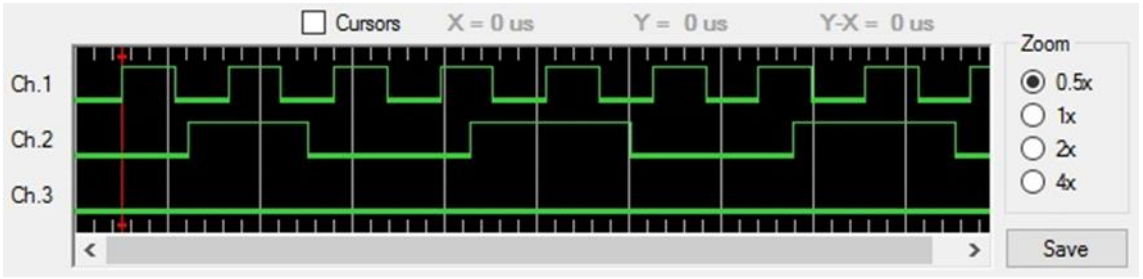


Figure 10. Wrong Sequence - FF based implementation

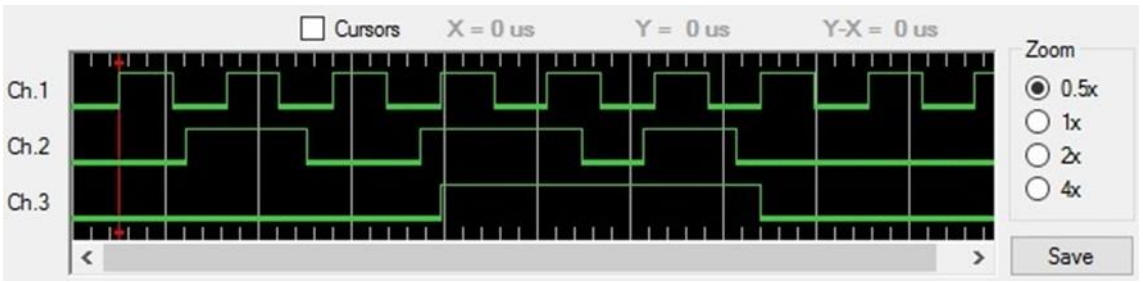


Figure 11. Right Sequence - FF based implementation

In figure 10, the "101" sequence is not present at the input so the output is always 0. In figure 11, the "101" sequence gets detected (it starts at the second rising edge of clock) and the output is turned to high until a low level at the input is detected.

Figures 12 and 13 show the signals for the second implementation. Again, channel 1 is the clock, channel 2 is the input and channel 3 is the output.

In figure 12, the "101" sequence is not present at the input so the output stays at 0. In figure 13, the "101" sequence gets detected (it starts at the fourth rising edge of clock) and the output turns to high until a low level at the input is detected.

Conclusion

In this application note, two versions of a binary sequence detector were implemented using the capabilities of the GreenPAK ICs. In the first implementation, flip-flops and LUTs were used, while in the second one, an ASM, Flip-Flops and LUTs were used.

It's important to note the synchronous use of the ASM with the additional flip-flops. If the binary sequence is short (as it is in this example), a Flip-Flop based implementation can be simpler than one based on the ASM. For more complex designs, using the ASM may be more efficient.

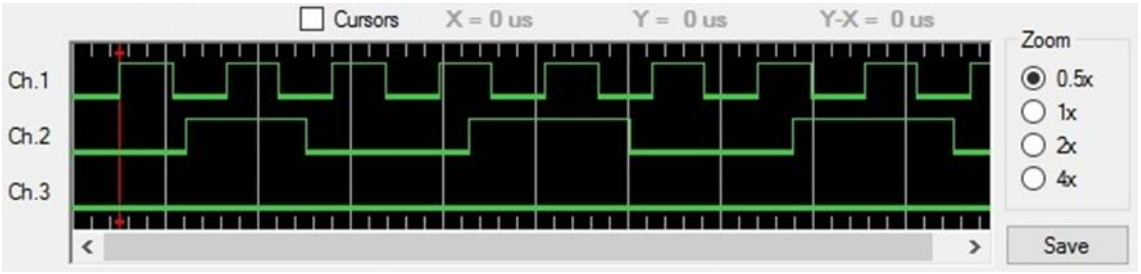


Figure 12. Wrong Sequence - ASM based implementation

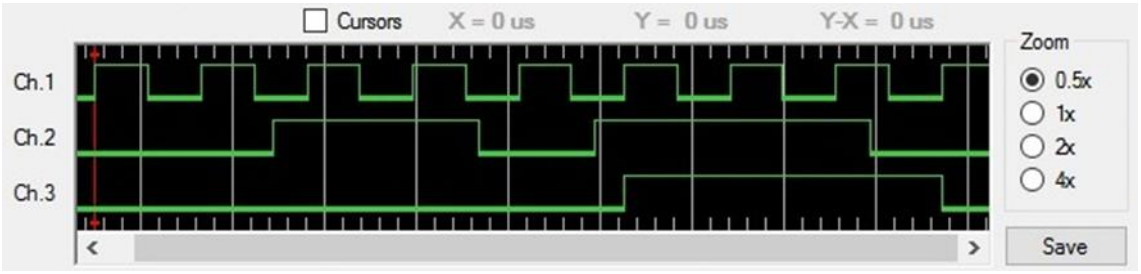


Figure 13. Right Sequence – ASM based implementation

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers skilled in the art designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only for development of an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising out of your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Rev.1.0 Mar 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.