N- BIT ADDER

Hochschule Ravensburg- Weingarten

System on Chip, Prof. Dr. rer. nat. Markus Pfeil

Team Members:

Amoussou Z. Kenneth (33972)

Mahenth Jayakumar (33902)

Nitinchandra Hegade (33906)

Rushikesh Munde (34499)

Srinivasan Balakrishnan (33903)

Hatem Ahmed (34588)

Contents

1. Task Introduction:

The main objective of this task is to create program for an 8-bit adder circuit which will be implemented in the Cyclone CYC1000 microcontroller. Three interfaces on the microcontroller are used for this task they are as follows:

1. Input push button – To receive the input from the user
2. LED – To display the output
3. Reset button – To reset the process

The program will be implemented using state modelling approach and the function of this program will be tested using test bench.

2. Overview of the state diagram:

There are overall 5 states in the 8-bit adder program enumerated as follow:

1. INIT_STATE
2. IDLE_STATE
3. DEBOUNCE_STATE
4. READ_BIT_STATE
5. OPERATION_STATE

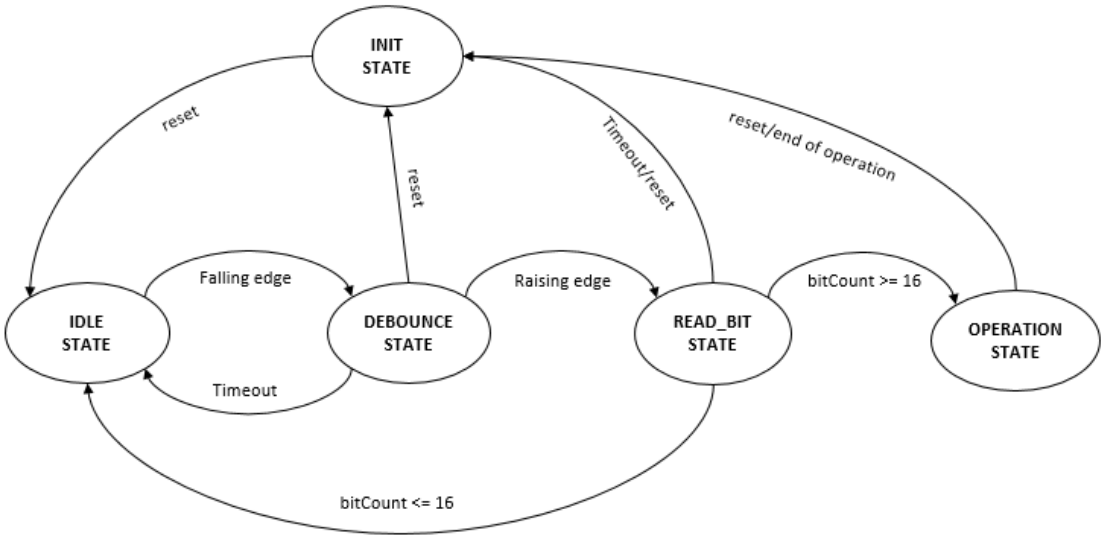

Figure 1. State diagram of Adder

After initialization, the system stays in IDLE_STATE and waits for an input from the user. On the falling edge of the input data line, the system enters the DEBOUNCE_STATE. Debouncing the input data line is quite important as it'll be attached to a button and an undebounced button input could lead to flickering or unexpected behaviour of the system. So, the debounce state mainly consists in waiting for a "DEBOUNCE_TIME" and checking back again that the data line is still low. When it's not the case, the system just goes back into IDLE_STATE. However, if the input data line is still low after the debounce time, the system moves into the READ_BIT_STATE. As we only have a single input to provide both 0-bits and 1-bits, a time threshold has been defined to differentiate between a 0-bit input and a 1-bit input. In the READ_BIT_STATE, if the input data line goes high before the time threshold defined, the input is considered as a 0-bit. However, if the data line goes high after the time threshold, the input is considered as a 1-bit. By tracking the number of bits already provided by the user, the system moves into the OPERATION_STATE when all 16-bits have been provided. After the OPERATION_STATE, the system must be reinitialized via the reset input.

3. Program Description:

The programs structure is now in three parts:

1. In the first part of the library, entity and architecture are defined. The standard libraries required for this program are used. In the entity part "clockFrequency" and "size" is defined in "generic" section. The various timing variables such as BIT_READING_TIME, DEBOUNCE_TIME and INPUT_TIMEOUT is initialised from clockFrequency variable and the "size" variable is used to define the number of bits of the adder. The "port" section of the entity contains the variables for the input button, reset button, output led and clock. Now the "architecture" part starts with the state enumeration and the timing variables are defined. The input from the user has been stored into a single variable named "operands" then it is split into "A" and "B" (using alias), operands is 2 times the bit-width of the adder. The following signals are included in the architecture namely: bitCount, timer, currentState and the nextState.

2. The second part of the program contains the process which handles the reset, and clock inputs as well as the timing function of the state machine. The reset button is defined as a low- active input. If the reset input is activated the timer and the state variables are reinitialized. The timer variable is incremented on every rising edge of the clock pulse when it is inside a particular state and it gets reset when the state transition happens. This way of handling the timing allows us to eliminate the individual time variable for each state.

3. The third part deals with the state handling process. currentState, input and timer are in the sensitivity list of this process. As mentioned earlier there are five states which will be executed sequentially based on the user inputs. The state and bitCount variables are initialized in the INIT_STATE. Since input is low active the idle state checks for the falling edge of the input. Once the input is received it moves to DEBOUNCE_STATE where the timer increments until it reaches a defined debounce time. If the button is still pressed, it moves to READ_BIT_STATE or it will move to IDLE_STATE. In the READ_BIT_STATE, the number of bits already read is checked via the "bitCount" variable. If all bits of the operands have been read then the state machine moves into OPERATION_STATE. Otherwise, it waits in that state. This bit count checking is done at the beginning of the READ_BIT_STATE to avoid array indexing error in the operands variable. In case the bitCount variable is less than the size of the operands then the process checks for the rising edge of the input which indicates that the user has released the button. At that moment it checks for the number of clocks ticks the input has been held "low" after the debouncing step (total time the input button is pressed). If the number of clock ticks is less than the BIT_READING_TIME then the corresponding bit is set to bit “0” otherwise it is set to "1". The same process is repeated for the consecutive bits by incrementing bitCount variable after each bit value assignment. If the process doesn’t see a rising edge of the input till INPUT_TIMEOUT which indicates that the user doesn’t release the button till the defined timeout, then the process moves to INIT_STATE where variables get reset. Once all bits are read, the process moves to Operation state where two variables “A” and “B” (These are the first and second half of operands Variable) are get added and the result stored in output variable.

4. Test bench Description:

Testbench has two parts: in the first part, required variables and clocking process are defined and in the next part an algorithm is used to check the output of the adder in multiple test cases.

1. The testbench starts with the definition of standard libraries, then its architecture is defined. In architecture, the generic variables i.e., size and frequency are assigned with values. Since this is an 8-bit adder, size gets the value of 8, and frequency is set to 1KHZ. The “T” variable is used to create clock pulses, which is set to inverse of frequency i.e., 1ms. Followed by these generic variables, the signals input, output, clock and reset are defined. Architecture contains three signals namely testtable, tb_sum and encoding which is used to test the behaviour of the adder for various input cases. These input cases are defined in testtable which is of the type TESTBENCH. The values stored in testtable are processed and provided as input to the adder one bit a time. The output of the adder is compared with the "tb_sum" variable. The encoding signal is used to test the adder for short and long encoding cases. After signal assignment, the port mapping is done with input, output, clock and reset variables. The clocking process is used to create clock pulses for 1ms.
2. The second part of the testbench involves test logic which test the adder for all input cases defined in testtable. In these logic two loops are defined. In the outer loop the process goes through each input cases one by one till the length of the testtable. On each case, the adder operation is checked by the inner loop. The inner loop goes through each bit of the input case. Based on the input signals, an encoding is used to simulate different duration of button press as described in table 1.

3. *Table 1: Button press timing for Short and Long Encoding*

Encoding	Bit	Button Press Timing
Short (0)	0	8
Short (0)	1	23
Long (1)	0	19
Long (1)	1	40

Since the debounce time is 5 ms and bit read time is 15 ms. The adder should produce zero if the button press time is within 6 to 19 ms and produce 1 if it is within 21 ms and the input timeout. The encoding signal is used to simulate the short and long button presses. If the bit in encoding signal is 0, then the short button press is simulated (8 and 23 ms) and if it is 1 then the long button press is simulated (19 and 40 ms). The adder should produce the correct output in both the cases. Similarly, each bit of the input case is given to the adder and the result of the adder is obtained in output variable. This completes the inner loop. An assert statement is defined at the end of the outer loop which compares the output with tb_sum to check the correctness of the solution for each input case. Once all cases are checked by the outer loop, an assert statement notifies us with end of simulation.

5. Simulation Result:

The simulation is done for all the input cases in the testtable. Totally 16 input cases have been tested. Below you can find the whole simulation graph which spans for about 9357ms. The initial case is “0101000000011001” and the final case is “1111111111111111”.

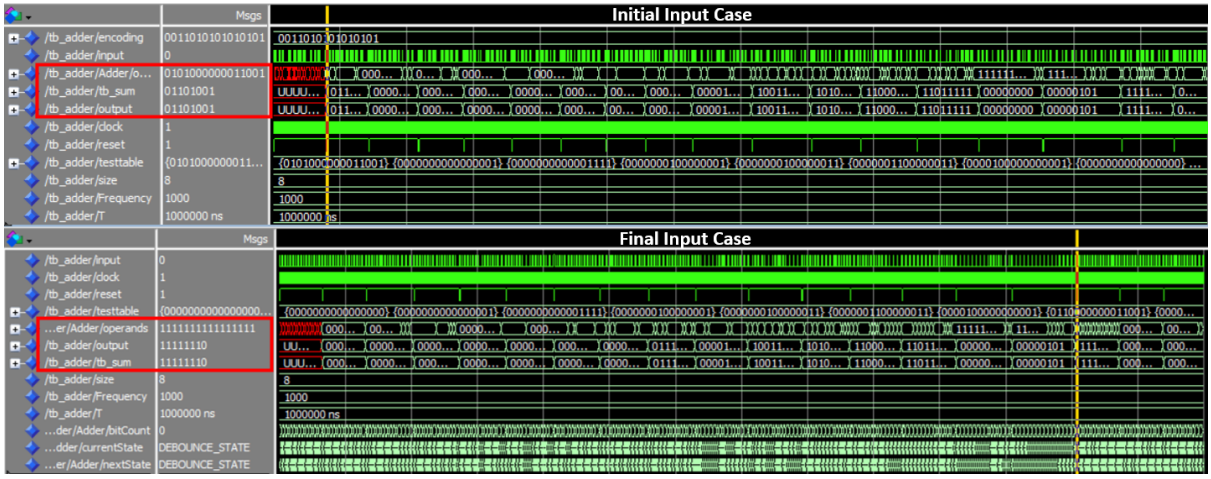

Figure 2. Simulation result for all inputs in the testtable

The picture below explains the functionality of the adder for one of the input cases. The first part shows a short encoding wherein the correct bits are produced for 8 and 23 ms button presses. In the following part, the long encoding is shown for 19 and 40 ms wherein the correct bits are produced by the adder.

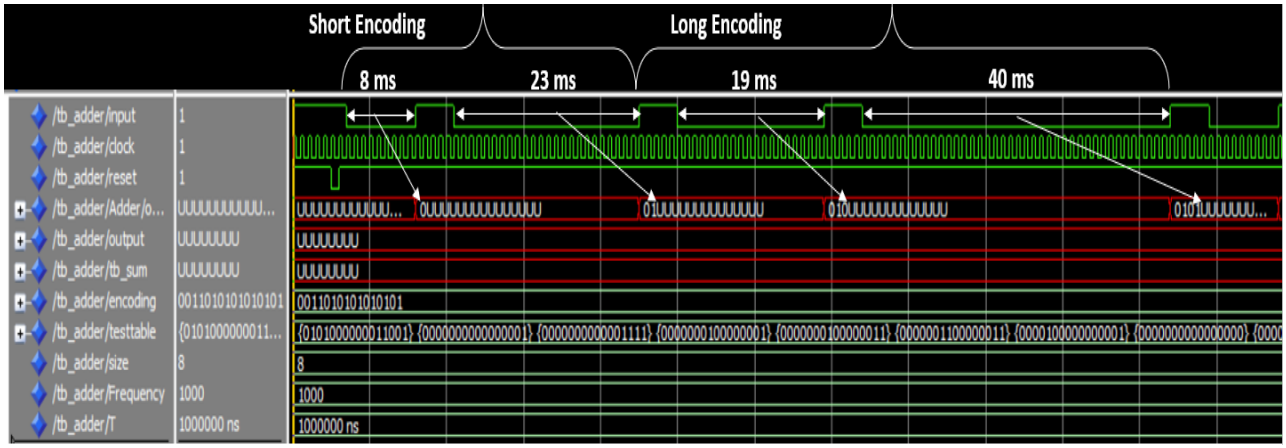

Figure 3. Bitwise short and long encoding simulation

The picture below shows the state wherein the input is held low and never released. In this case once the INPUT_TIMEOUT has been expired the state machine goes into the IDLE STATE.

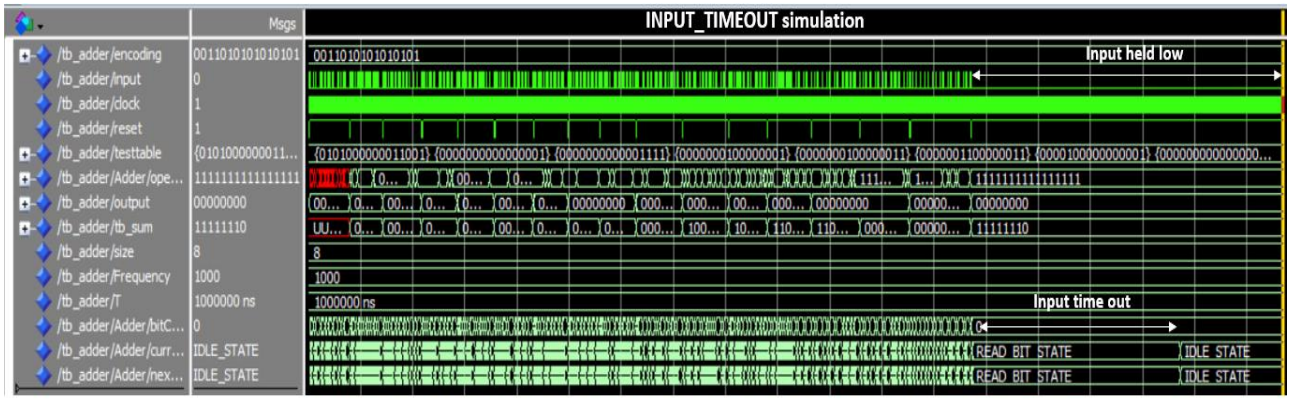

Figure 4. Input time out simulation

5.1 Features of this program:

- The main advantage of the method that has been used is the flexibility of the adder obtained. It is possible to synthesize the adder easily for any bit-width by setting the size parameter according to the expected size of the adder.
- The testbench provided checks the output of the adder for expected result in order to stop any issue in the data reading process. During the simulation, if at any stage of the test input data, the output of the adder doesn't match the expected result, the simulation is automatically stopped; and an error message is displayed.
- The testbench is constructed in such a way that it can test multiple input and encoding cases which greatly increase the reliability of the output produced.

6. Conclusion:

Knowing that the system has a single data input, a simple time-based protocol has been defined to acquire and process the user's inputs. The different timeouts defined prevent the system to be locked permanently into a given state. The flexibility offered by the method allows to synthesize the model for an X-bit adder without the need to change the architecture definition.