# Button Controller

The DE-01 SoC Development Board has multiple manual inputs, including buttons and switches, to provide an interactive interface. In this project, four of four push-button inputs and three of nine switches are used. The SW1 and SW2 are the mode selection switches, while SW0 is the Button Management switch, explained in this section. This controller aims to manage debouncing and synchronization of the inputs and provide necessary state and pulse outputs used in other sub-blocks and the state machine.

## DE-01 SoC Development Board Input Schematics

To design a proper controller, first, the input characteristics should be analyzed carefully. The schematic file is provided in a [GitHub Repository](https://github.com/sahandKashani/SoC-FPGA-Design-Guide/blob/master/DE1_SoC/Documentation/DE1-SoC%20Schematic.pdf). The four push-button sections is shown below.

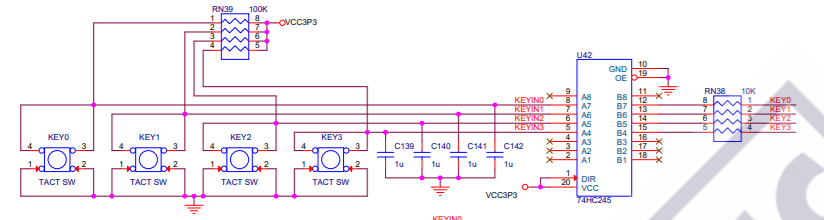
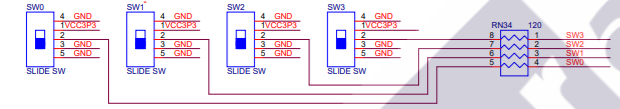


Figure - Push Button Schematic Section

There are multiple essential points that should be extracted from the above schematic. The first property of the input is ACTIVE LOW since a button press pulls the KEYIN# to GND. In addition, the IDLE state of the buttons is HIGH, due to 100K pull-up resistors. Another critical point is having a 1uF bypass capacitor connected to KEYIN# for each pin, preventing high-frequency content or noise from affecting the output state of the button. A simple calculation of shows that there is a strong bounce filter. However, an addition of software debouncing applied to satisfy more generic design, which can be used other board not having these bypass capacitors connected, still runs correctly. 74HC245 IC is the octal-bus having tri-state buffers and direction selection for general purpose design; but, as it can be seen from the above figure, this 74HC245 is fixed direction and has no High-Z output feature, only transfers left side to right side combinationally.



Switch connections have no external capacitors, only serial resistors to limit current; therefore, these inputs are vulnerable to mechanical switch noises. Therefore, these inputs also cleaned using debounce modules.

## Debounce Module

The mechanical inputs have a common problem of mechanical switch bounces and arcs between connections resulting in fast state transitions, which can confuse the logic and state structure if not appropriately handled. The module has a characteristics structure containing a counter and a rising or falling edge detector. The main idea behind the software debounce module is to check that input is stable for a given amount of time. If the input alters before the given time-out, the timer starts over from zero again. The disadvantages of using the software debounce module are the necessity of increasing of registers due to separate counters for each input and the fixed amount of latency occurs from the actual state transition. On the other hand, the absence of this filter can lead to considerable errors in mission-critical processes.

The debounce module in the project design has two versions for push-buttons and switches separately since one is active low, the other is active high. The active-low push-button outputs are also converted to active high in order to provide more logical and intuitive output for other sub-block and logic structures.

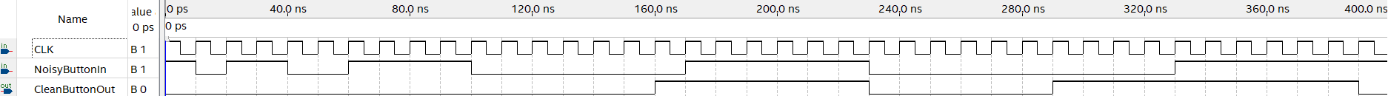


Figure - Button Debounce Simulation Results

The idle state of the push button is High, and for practical purposes, counter is set to count to max 4 to observe reasonable outcomes. Therefore, any four consecutive bits trigger the state transition of cleaned output. However, if 100ns time instant is inspected, it can be seen that the button is pressed (pulled low), but six clock latency is observed instead of four. This is because the synchronization registers synchronize the input to the system clock and edge detection logic. In real life, since the counter limit is up to , these two synchronizations register effect can be easily neglected. Moreover, while the input is active low, the output is converted to active high to represent button press. Counter limit checks the input whether it is still for to decide it is a stable input. This limit can be adjusted according to the noise characteristics of the buttons.

## Button Manager Module

The DE-01 SoC Development Board has four user buttons to interact with; however, the project requires more than four distinct triggers for digit inputs and direction inputs besides command inputs like select, cancel, etc. Therefore, this module allows the user to switch between push button roles between informative inputs and command inputs using the SW0 switch. When SW0 is high, push-buttons are in the command mode representing select, cancel, backspace, or other state-depended actions. While SW0 is low, push-buttons are representing either digits or direction keys depending on the shopping state. The active high current status of both KEY and CMD information is kept in KEY\_Reg and CMD\_Reg registers, respectively, to inform other blocks of the current statuses of input in the level form. The active-high level form can be used in trigger or condition for logic circuits.

## Button Level Pulse Convertor

An active-high level status of a button can represent whether the button is pressed or not at the time instant when the check is done. However, if an enable-like input needed to be driven according to the time instant when the debounced clean button pressed action is received, a pulse should be generated for one clock period. This pulse signal can trigger shift register modules to take only the specified input by the state machine to save. Since the one-clock-period pulses are helpful for such operations, Button Controller must provide these for each CMD\_Reg and KEY\_Reg statuses.

## Button Controller Extensive Functional Simulation Results

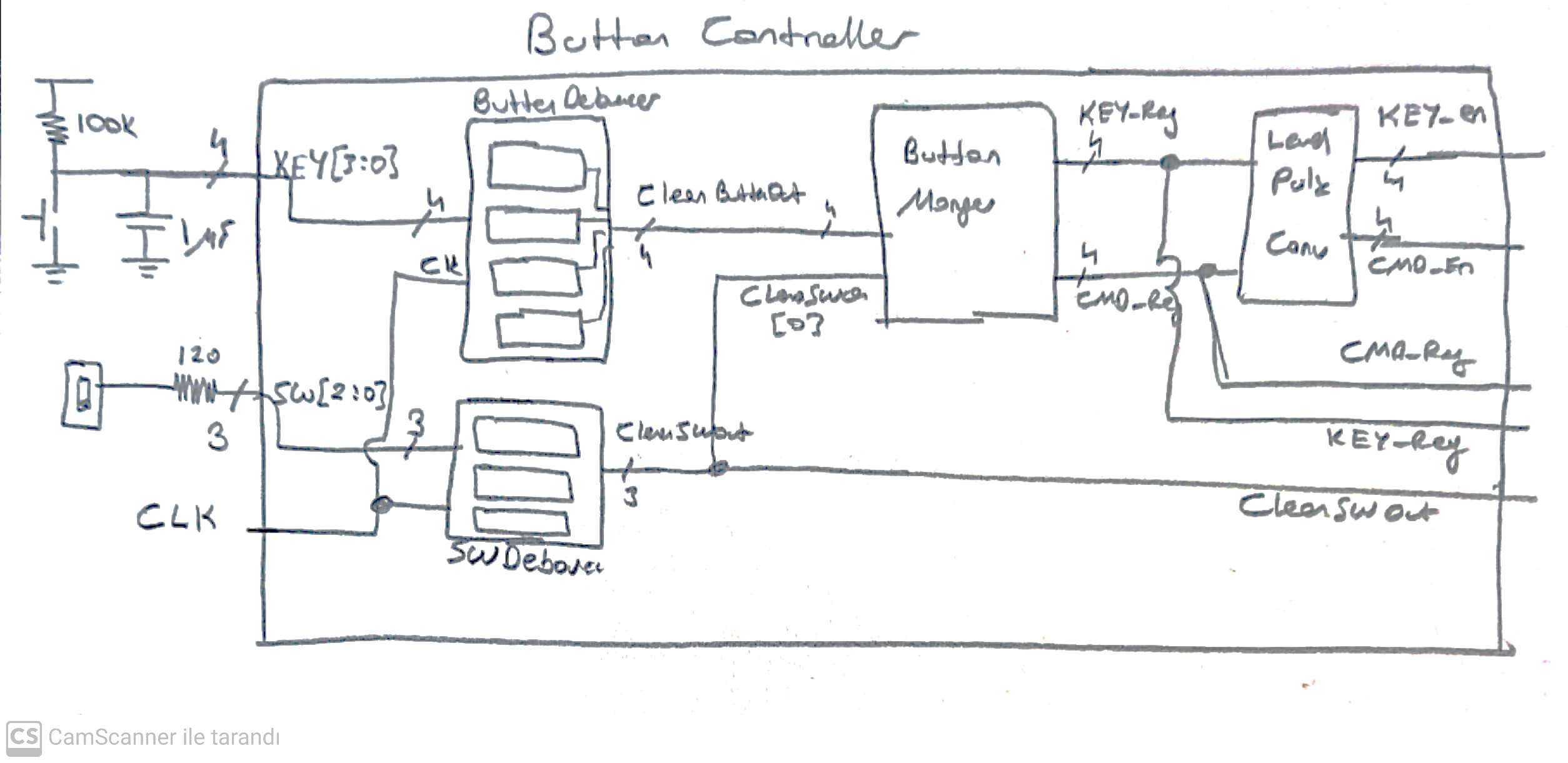
The overall ButtonController structural block diagram is shown below for clarity. 

Figure - ButtonController Structural Block Diagram

Button and switch electrical characteristics are added to give proper understanding.

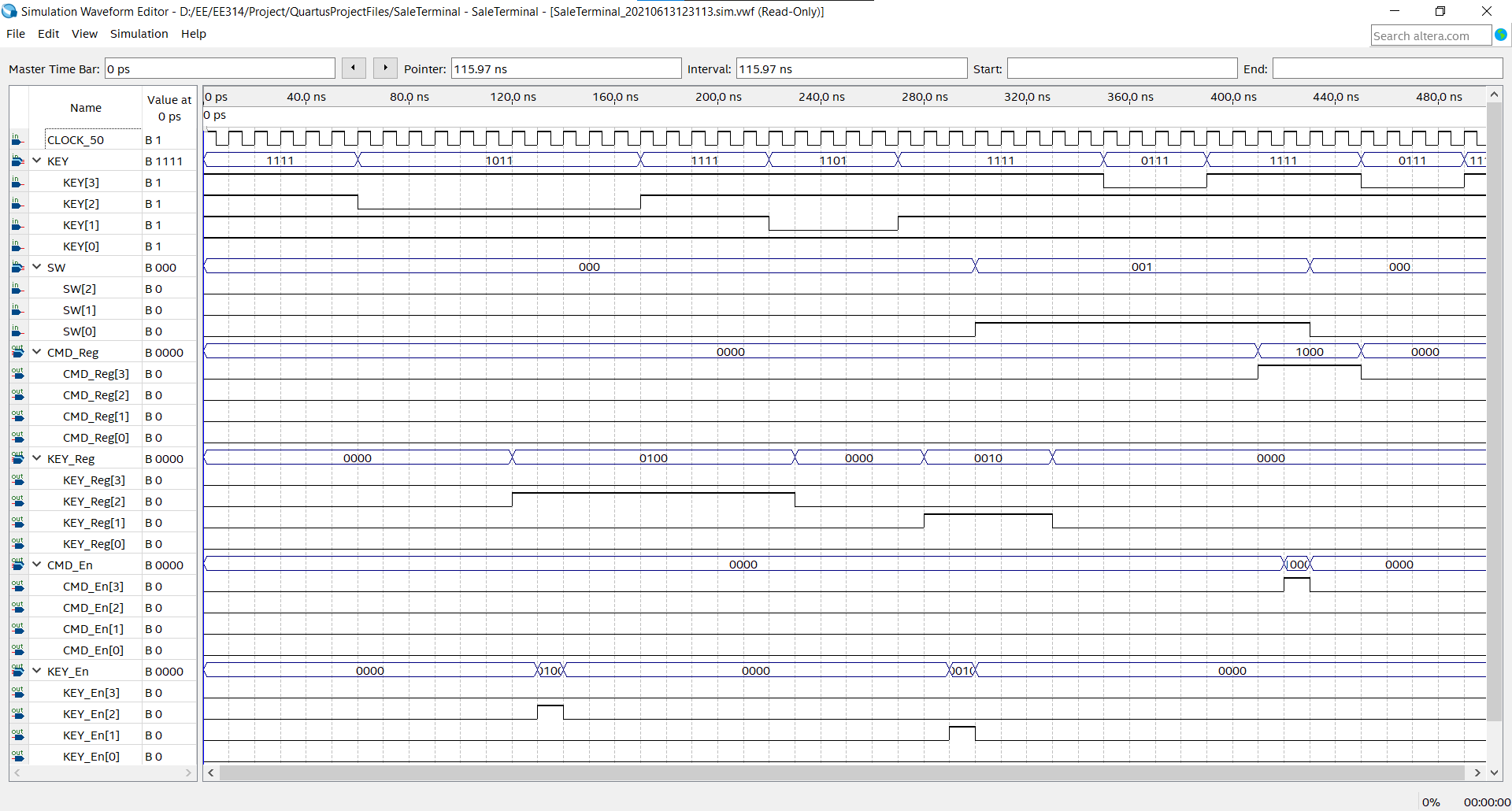


Figure - ButtonController Functional Simulation

This simulation shows multiple details and sub-block results in the same test; therefore, careful inspection is recommended. The debounce module counter is again decreased to a 2-bit register to observe reasonable outcomes. A remind: KEY inputs are active low signals.

At 60ns time instant, KEY[2] is pressed, which is proceeding to debounce module combinational, then the ButtonManager module decides that this is a KEY (informative input) according to the state of SW0 and updates the KEY\_Reg[2] to the HIGH state after six clock cycles (120 ns)as expected. The State Machine now can provide necessary internal signals for sub-blocks which can be enabled using KEY\_En[2] signal followed exactly after one clock cycle to give time data for setup stack.

At time instant 300 ns, SW0 switches its state to CMD mode, indicating that KEY inputs are now representing CMD inputs. While SW0 is in CMD mode, at 350 ns, KEY[3] is pressed, and debounce module passes this input to ButtonManager module, since it is at least four clock cycle long (2-bit counter), and after six clock cycles after the actual key pressing, CMD\_Reg (SW0 High / CMD Mode) is updated. Furthermore, again a pulse signal is generated after one clock cycle.

# Hover Controller

The Hover Controller block is responsible for highlighting both products listed in the basket section and the product images. The product images have a square mark on the left upper side to indicate the product is selected. There are two conditions where the products are highlighted; first, every barcode digit entered triggers matching items; second, when the user is in the interactive selection mode. On the other hand, when Basket Edit mode is activated, only the product list is hovered, indicating the currently selected product. Hence, this controller consists of two distinct sub-blocks, and one is responsible for barcode-related hovering, the other is for selected item hovering depending on the operation mode.

The selected item hovering process is simplified since all other blocks are compatible with Product IDs. Either Basket Edit or Interactive Select Mode, only one selected item has hovered; therefore, we use either array ID or Product ID in our algorithm. On the other hand, the matching barcode highlighting process is more complicated and resource-intensive. The Barcode Hover Subblock should be able to identify possible products according to the current state of the barcode entered. Although the task seems like a database search structure in the programming aspect, it can be more efficient when it is considered a hardware description language. Therefore, we manage to generate a possible product list with no register used in this subblock. This optimization resulted in freeing significant available resources that we can use for other operations.

# State Machine

The state machine is responsible for managing other modules depending on the operation mode or state. It sends activation, enable, and reset pulses when needed to operate correctly. There are seven states, namely, “Start”, “Idle”, “Barcode”, “Interactive”, “Quantity”, “BasketEdit”, “EndShopping”. These states are based on the nature of the design. “Start” state is for preparing the modules for new customers and resetting previous information. When reset is done “Start” state directly goes to “Idle” state, where the operation mode is decided and redirected. There are three operation modes, “Barcode Selection Mode”, “Interactive Selection Mode”, and “Basket Edit Mode”. According to mode prerequisites, the “Idle” state redirects the state to the proper mode. The “Idle” state handles the transition between operations. Basket Edit operation mode is superior to other operation modes; when activated, it has priority. The condition for every operation mode is checked inside its state; hence, a transition between operations can be catch and handled by again “Idle” state. After any product selection by either Interactive or Barcode, the state is transited to “Quantity” state, where the machine waits for quantity input from the user or the cancel action. When quantity is entered, the state machine handles the other controllers in order to add the product to the basket and display it. Then, it again forwards the state to “Idle” state where the operation mode is decided at the next clock pulse, reaching either “Barcode”, “Interactive” or “BasketEdit” states.

On the other hand, if “BasketEdit” state is reached, the state waits for direction input from the user. If the user wants to select the item which is wanted to be deleted, “Select Command Button” is pressed. Finally, when the user wants to end the shopping, “End Shopping Command Button” is pressed, which leads to “EndShopping” state. The “EndShopping” state waits for the user for confirmation by pressing “End Shopping Command Button” again. When the confirmation is done, it directs the state to “Start” state, where a new shopping environment is prepared.

In figure x, the State Machine Viewer result generated from Quartus Prime can be seen. For a detailed state flowchart, please refer to Appendix x.

