

Tunku Abdul Rahman University College

BAAP 2104

Virtual Instrumentation

***Laboratory Practical #7***

***Human Interface Device (HID) Control***

Foong Ghin Yew

(15 WAU 09184)

14th July 2016

**Objective:**

The objective of the practical is to control a Human Interface Device (HID) by displaying the keyboard input sequence, the input hold time and activate a Beep Sound when the input hold time exceed 1 second.

**Introduction:**

Human Interface Device (HID) is an important device that allow human to give input commands and instruction to the computer for subsequent execution. Keyboard, mouse and joystick are examples of HID. One of the important skill for equipment and instrumental control is to capture HID input and program its outcome. HID signal captured is always auto sorted by default by the input register OS. Therefore, this practical will un-bubble sort the input to follow the First In First Out order. Next, key pressed for a long time could cause trouble to the user. Thus, this practical include alerting user when the key pressed for more than a second by generating a ‘Beep’ sound.

**Procedure:**

1. The system was broken down into a main-vi and two sub-vi namely GetNewKeyPressed and CustomBeep respectively.
2. GetNewKeyPressed compare two array with size of n and n+1 then extract the extra +1 element that was not in the n-array.
3. A frame structure with two frames were added to GetNewKeyPressed sub-vi where the first frame initialize all the output value.
4. For Loop was used to check every element in the n+1 array in the second frame.
5. Search 1D Array was implemented to check if the element in each cycle of the For Loop was in the n-array.
6. When extra element found, outputs ‘Changes?’ was set to true, ‘Extracted different’ as the extra element found and ‘Extract Index’ set as the index of the extra element in the n+1 Array.
7. The other sub-vi, CustomBeep configured the Beep.vi to use custom frequency of 1200Hz with a duration of 100ms.
8. A Case Structure then covered the Beep.vi configuration in the True case only.
9. Comparator Greater Than was used to control the case structure by compare the input hold time and the input Threshold to Beep.
10. A big While Loop was implemented in the main-vi consisted a delay of 1ms that make sure each cycle of the loop was 1ms.
11. Initialize Keyboard.vi was used to obtain the device id and Acquire Input Data.vi was used to get the input key pressed.
12. Close Input Device was then used to free the device from using in the main program.
13. The input key pressed connected as the n-Array of GetNewKeyPressed and the n+1 Array of another GetNewKeyPressed respectively.
14. Two empty array was created outside the loop to initialize the Sequence Array that will be shifted in each cycle of the loop.
15. One of the external array was used to record the time whereas the other used to store the input key pressed in sequence.
16. Two GetNewKeyPressed.vi were used to obtain the new key pressed and to obtain the key released individually.
17. Build Array was implemented to add new key pressed element found to the Sequence Array and add a ‘0’ to the Timer Array.
18. Delete from Array was be used to remove the element using the ‘Extracted Index’
19. Custom Beep was implemented to compare the first element in Timer Array and a constant of 1000 indicating 1s.

**Result Data:**

Sub-vi GetNewKeyPressed takes 2 inputs array and 3 outputs consisting an floating point integer ‘Extracted Different’, an integer ‘Extracted Index’ and a boolean ‘Changes?’ as shown in Figure 1. GetNewKeyPressed compare the two array and search for an extra element in the n+1 array. When an extra element was found, boolean output ‘Changes?’ was set to true. As shown in Figure 2 below, the element ‘2’ was not in n Array but contained in n+1 Array. Therefore, the ‘Extracted Different’ was set as ‘2’ and the ‘Extracted Index’ was set to 1 as element ‘2’ have index of 1 in the n+1 Array.

E:\Yew\USBEmerald\LabVIEW\HID\GetNewKeyPressc.png

Figure 1 Icon of sub-vi, GetNewKeyPressed

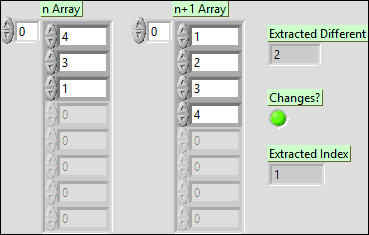


Figure 2 Front panel of sub-vi, GetNewKeyPressed

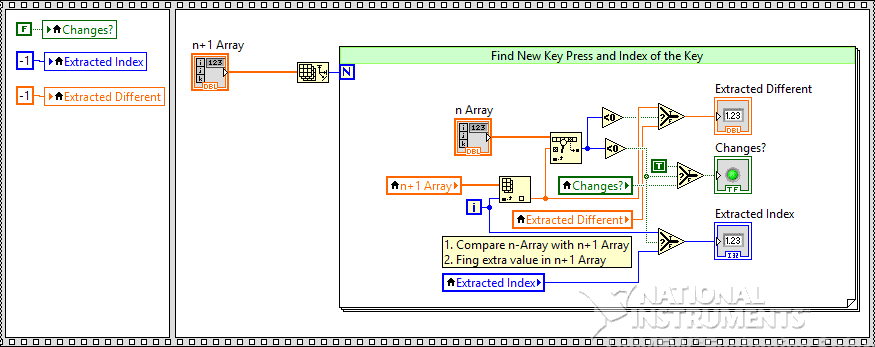


Figure 3 Block Diagram of sub-vi, GetNewKeyPressed

Sub-vi CustomBeep take two input of floating point number, ‘Threshold to Beep’ and ‘Holding time’ as shown in Figure 4. This vi compare the two input and activate a Beep Sound at 1200Hz for 100ms when the input ‘Holding time’ greater than the ‘Threshold to Beep’. As shown in Figure 6, this vi will do nothing if the comparison result was false as the false case was empty.

E:\Yew\USBEmerald\LabVIEW\HID\CustomBeepc.png

Figure 4 Icon of sub-vi, CustomBeep

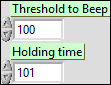


Figure 5 Front Panel of sub-vi, CustomBeep

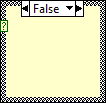
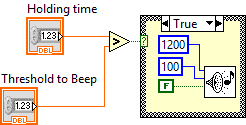


Figure 6 Block Diagram of sub-vi, CustomBeep, does nothing in the false case block

The main-vi namely UnsortAndBeep used the two sub-vi revealed previously to unsort the Key Pressed sequence and generate Beep sound. Figure 7 below shows the result of unsorting the key pressed. Keys Pressed array are acquired and sorted directly from the HID (keyboard) as shown in Figure 7. The right side shows the First In First Out ordered key pressed in Sequence array and its respective hold time in Timer array.

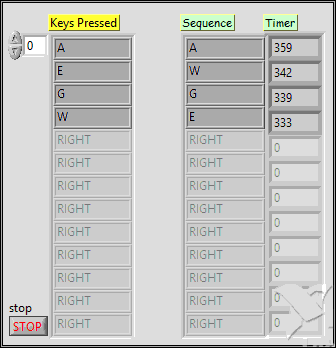


Figure 7 Front Panel of the main vi that display the key pressed in sequence and the hold time

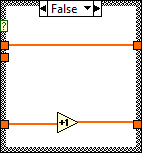
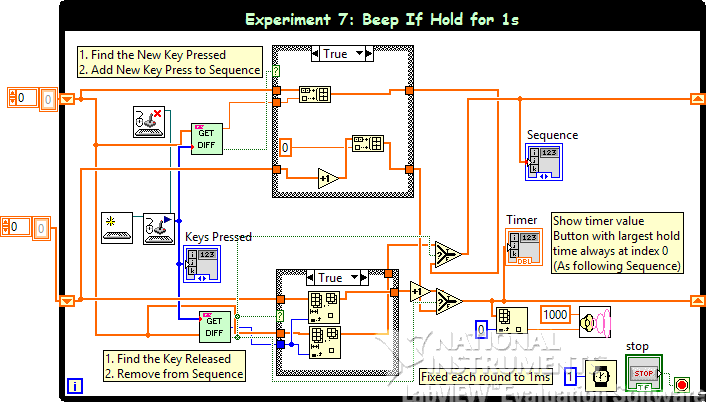
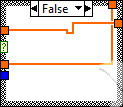
 

Figure 8 Block Diagram of the main-vi, upper false blocks increasing the timer value whereas lower false block do nothing

**Analysis & Discussion:**

This practical used pooling method to as to update the timer display every single milliseconds. A delay of 1 millisecond was shown in the bottom right of Figure 8 above to make sure each cycle used a millisecond to execute. In Figure 8, beep sound only check for the first element in the Timer array because the key pressed with the largest hold time will always stick at index 0. This is because both Sequence array and Timer array were unsorted and followed the First In First Out order. The first key pressed (First In) will be added to the array first and therefore must be having the largest hold time compare to the other keys. However, the disadvantage of the system was that the precision of the hold time will be limited to 1ms.

There was one critical issue needed to take care which is to remove the released key from the Sequence array and Timer array when the Key Released. If the released key was not removed from these arrays, the hold time will continuously increase and caused the beep sound to continuously sound.

There was a thought that the program may execute longer than a millisecond which may pollute the timer count using the delay. However, after some experiment by changing the delay value and display value, the delay was considered consistent and therefore remained at 1 millisecond. Another issue then rise at the CustomBeep.vi. As the beep sound was configured to a duration of 100ms. The timer count may goes inaccurate as each cycle now consist of approximately 100ms instead of 1ms. This issue can be solved by stressing the duration of beep sound to 50ms and increasing the delay timer to 50ms instead of 1ms. This action makes each cycle 50ms and therefore the Timer array values will have to multiply by 50 before displaying on the front panel. This issue can also be solved by using Event Structure by using a timer interrupt as the event to beep the sound.

**Conclusion:**

As a conclusion, inputs of a keyboard can be unsorted by registering the each key pressed into an array and check for the existence in the array. Hold time of the key pressed can be counted by fixing the delay time in each cycle of the While Loop and increment the hold time value by 1 in each cycle. Adding the key press to the array was important but removing the element when the key released was equally important as to stop the hold time count and also to arrange the sequence correctly.