Labor 1: Eingaben über eine Hexadezimal-Tastatur

Semester/Gruppe/Team: 04 / 01 / 06	**Abgabedatum:**	**Protokollführer:** Thomas Schlundt
Versuchstag: 17.04.2019	25.04.2019	**weitere Versuchsteilnehmer:** Yassine Tourki Kokou GAITU
Hochschullehrer:		
Kommentar des Hochschullehrers:		

Inhaltsverzeichnis

1 Einleitung

In diesem Praktikum wird eine Hexadezimaltastatur von einem Mikrocontroller ausgewertet. Dafür wird als Vorarbeit die Signallaufzeit der Tastatur und die Ausführungsgeschwindigkeit von forschleifen innerhalb des Mikrocontrollers gemessen. Da sonst das Eingangs Signal zu schnell gelesen wird und das falsche Signal ausgewertet wird.

1.1 Geräteliste

Diese Geräte wurden im Versuch verwändet.

- Tiva TM4C1294NCPDT
- Tektronix MSO 2024 Tastkopf: Tek P6111B
- Hex-Tastatur des Labores

2 Ausführungszeitmessung mit dem Oszilloskop

In diesem Aufgabenteil wird ein periodisches Rechtecksignal an einem Pin eines Ausgangsports des Mikrocontrollers ausgegeben. Mithilfe der Verwendung einer for-schleife, konnte die Periodendauer des Rechtecksignals durch die Veränderung der Zähldurchgänge festgelegt werden. In dieser Aufgabenstellung musste die Verzögerungszeit für eine bestimmte Anzahl von Durchläufen mithilfe des Oszilloskops gemessen werden, um zu prüfen, ob es ein lineares Verhältnis zwischen der gemessenden Verzögerungszeit und der Anzahl der Schleifendurchgänge vorliegt.

2.1 Auswertung

Nun folgen vier Messungen mit *10 Abstand.

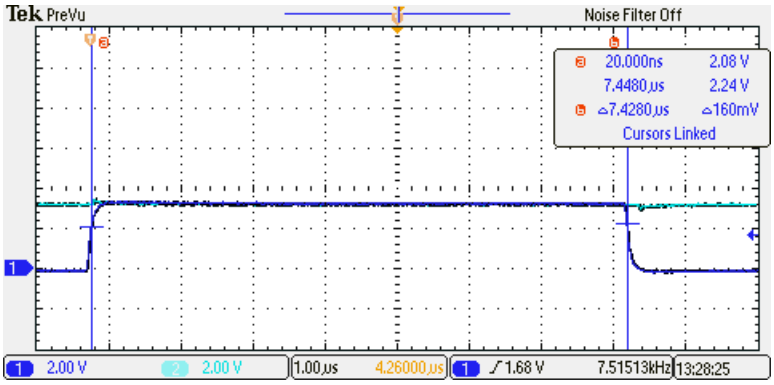


Figure 1: Schleifenanzahl = 10

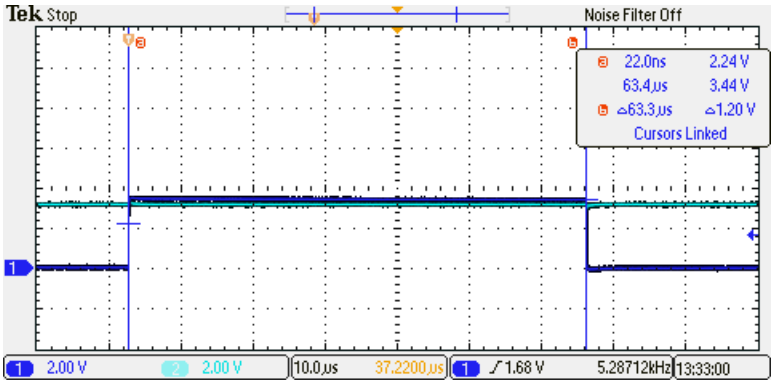


Figure 2: Schleifenanzahl =100

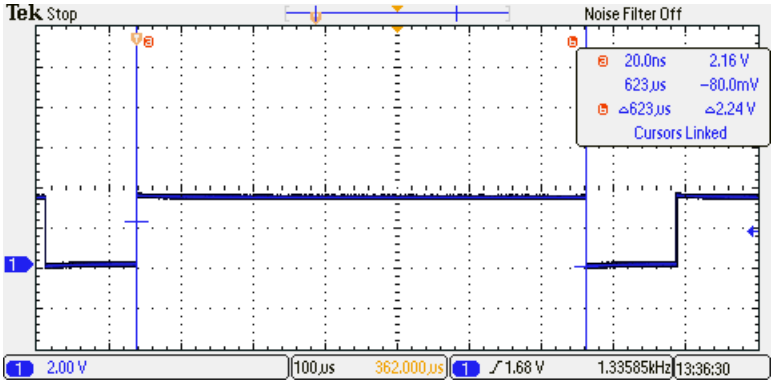


Figure 3: Schleifenanzahl = 1000

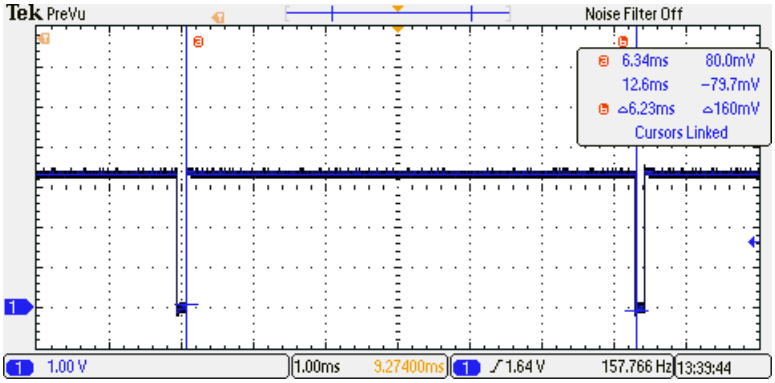


Figure 4: Schleifenanzahl = 10000

Table 1: Periodendauer für unterschiedliche Schelifendruchmesser

Anzahl der Durchläufe (Schleifen)	Verzögerungszeit [µs]
10	7,42µs
100	63,3µs
1000	623µs
10000	6230 µs

Aus der Tabelle heraus ist ersichtlich, dass die Schleifendurchläufe linear zur gemessenen Periodendauer sind. Die Warte-Methode(timeelapse)[1] wird benötigt, da das physikalische Signal vom Ausgang bis zum Eingang des Ports M verzögert erscheint. Wir können auch berechnen, dass die durchschnittliche Verzögerungszeit pro Schleifendurchlauf in etwa 0,7 µs beträgt. In der Realität bei nur einem gemessenen Durchlauf aber von dem Wert leicht abweicht.

3 Signallaufzeitmessung

Die Laufzeitverzögerung des Signales vom Ausgangspin des Mikrocontrollers durch die Hexadezimaltastatur bis hin zum Eingangspin des Mikrocontrollers wird ermittelt. Um bei der Messung zu wissen, ab wann der Mikrocontrollern einen High- oder Low-Pegel erkennt, schauen wir ins Datenblatt.

$V_{DD} = 3{,}3V$
Dem Datenblatt zu entnehmen wird ein High_Pegel bei bei $0{,}65*V_{DD}$ erkannt und ein Low-Pegel bei $0{,}35*V_{DD}$.
Der Mikrocontrollern erkennt einen High_Pegel ab 2,14V und einen Low-Pegel ab 1,15V.

Table 27-7. Recommended FAST GPIO Pad Operating Conditions

Parameter	Parameter Name	Min	Nom	Max	Unit
V_{IH}	Fast GPIO high-level input voltage	0.65 * V_{DD}	-	4	V
I_{IH}	Fast GPIO high-level input current [a]	-	-	300	nA
V_{IL}	Fast GPIO low-level input voltage	0	-	0.35 * V_{DD}	V
I_{IL}	Fast GPIO low-level input current[a]	-	-	-200	nA
V_{HYS}	Fast GPIO Input Hysteresis	0.49	-	-	V
V_{OH}	Fast GPIO High-level output voltage	2.4	-	-	V
V_{OL}	Fast GPIO Low-level output voltage	-	-	0.40	V

Figure 5: Auszug aus dem Datenblatt des Microcomputers, Seite 1820

3.1 Messergebnisse

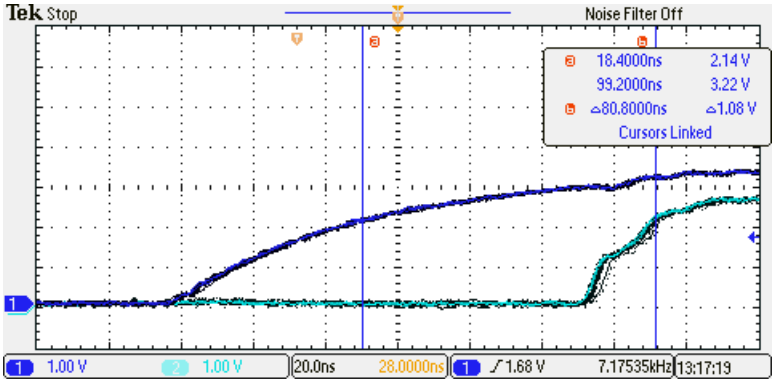


Figure 7: Steigende Flanke

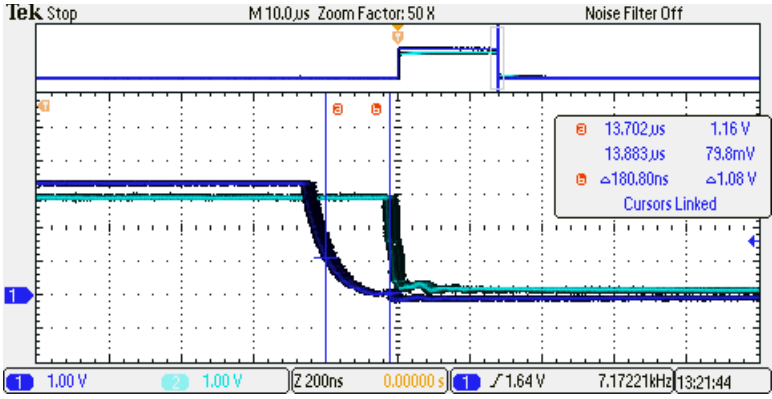


Figure 6: fallende Flanke

Die ansteigende Flanke ist um 80ns verzögert und die fallende Flanke um 180ns.

3.2 Auswertung

$$\frac{10000}{6.23\ ms} * 180\ ns * 2 = 578$$

Somit wird zwischen dem Ausgang setzen und Eingang lesen eine Warteschleife von 578 Durchgängen benötigt.

4 Hexadezimal Tastatur

4.1 Aufgabenstellung:

Es soll ein Programm für den Mikrocontroller geschrieben werden, welches ermöglicht die 4x4 Hex-Tastatur zu nutzen. Dabei sollen nur vier Ausgänge und vier Eingänge des Mikrocontrollers verwendet werden. Durch eine Musterfolge soll erkannt werden, welche Taste gedrückt wurde. Dazu soll bei einem langen Tastendruck die Taste wiederholt jede Sekunde in der Konsole ausgegeben werden. Außerdem soll eine Fehlermeldung erscheinen, wenn mehr als nur eine Taste gedrückt wurde.

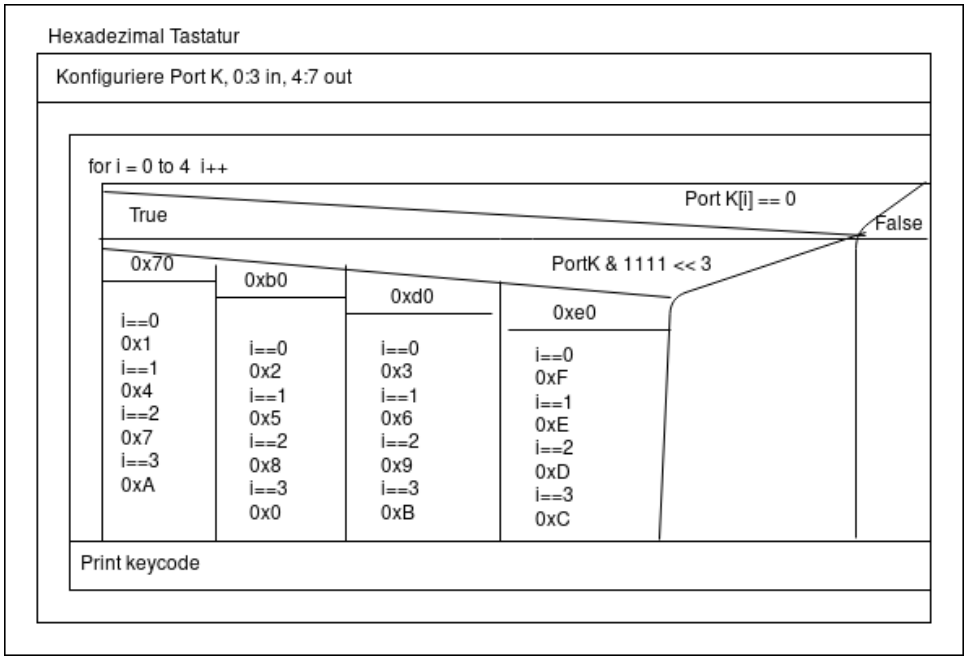
4.2 Lösungsansatz:

Das Programm ist in drei Hauptmethoden aufgeteilt. In der Methode **keyboard()**[2] wird zu Beginn die GIPOs des Microcomputers konfiguriert. Außerdem ist eine while schleife vorhanden, durch die das Programm niemals enden. In der while schleife wird die Methode **keyboardscan()** aufgerufen die einen Rückgabewert hat, die besagt welche Taste gedrückt wurde.

In der Methode **keyboardscan()**[2] ist eine For-Schleife die es ermöglicht das Ausgangsmuster zu schalten. Nachdem das Ausgangsmuster geschaltet ist, muss kurz gewartet werden und dann kann auch der Eingang wieder gelesen werden. Das gelesene Muster wird an die Methode **evaluation()** weitergegeben, welches schaut ob eines der ersten 4 Bits null ist. Trifft dies ein wird geschaut welches Muster es war und dadurch kann der Tasten Wert gesetzt werden.

Um zu überprüfen das nur eine Taste gedrückt wurde sind die Zähler countery und counterx zuständig. Der countery ist für die Spalten Zählung zuständig. Der counterx für die Zeilen, doch dieser Wert muss static sein damit dieser ein Muster Zyklus lang zählt. Wurde das Muster wiederholt, wird dieser Wert auf null gesetzt. Der Rückgabewert dieser **evaluation()** Methode ist bei keinem Tastendruck 0xEFFF und bei mehr als nur einem Tastendruck wird der Fehlercode 0xFFFF zurückgegeben.

Für die Tastenwiederholung wurde einfach nach dem Printen der Taste in die Konsole die Warteschleife eingesetzt in der while schleife der Methode **keyboard()**.



5 Fazit

Wenn mit einem Mikrocontroller gearbeitet wird, ist darauf zu achten, dass es zu Signalverzögerungen kommen kann. Es ist deshalb zu empfehlen diese Laufzeit zu ermitteln, um zu verhindern, dass Signale falsch aufgenommen werden. Bei Abfragen der Eingänge muss diese Laufzeit berücksichtigt werden.

Ein Problem könnte noch sein das die Zykluszeit sich ändert, wenn neuer Code hinzugeführt wird, da wir nicht den integrierten Timer mit Interup Funktion verwenden. So könnte es passieren das z.B. eine Taste länger gedrückt werden muss damit sie registriert werden kann.

Beim Programmieren bei der IF Abfrage wäre es besser lieber zu viele als zu wenige klammern zu verwenden, da wir einige stunden davor Saßen, nur um eine klammer hinzuzufügen.

Für eine elegantere Lösung habe ich im Internet [3] eine Möglichkeit gefunden bei der ein 2D Array verwendet wird. Zuerst wird ein Ausgang null gesetzt, dann die Eingänge wieder gelesen und wenn beim Eingang es an einer Stelle null ist. So hat man nun für den 2D Array eine Position und der Knopf wurde lokalisiert.

6 Quellcode

```
1.  #include "tm4c1294ncpdt.h" //https://tohtml.com/
2.  #include <stdio.h>
3.  #include <stdint.h>
4.
5.  #define COLUMNNUMBER 4
6.
7.  /*
8.          x1 = k7
9.          x2 = k6
10.         x3 = k5
11.         x4 = k4
12.         y1 = k0
13.         y2 = k1
14.         y3 = k2
15.         y4 = k3
16. */
17.
18. /// Pseudo Delay
19. void timeelapse(unsigned long timevalue)
20. {
21.     unsigned long i;
22.     for(i = 0; i < timevalue; i++);                 // wait until loop ends
23. }
24.
25. void rectangleSignal()
26. {
27.     SYSCTL_RCGCGPIO_R |= (1<<9);                // clock enable port D, must be done for every port
28.     while((SYSCTL_PRGPIO_R & (1<<9)) == 0);     // clock when port D is available, wait for port
29.     GPIO_PORTK_DEN_R = 0x01;                    // digital I/O pins enabled(activate)
30.     GPIO_PORTK_DIR_R = 0x01;                    // define output direction I/O pins
31.
32.     while(1)
33.     {
34.         GPIO_PORTK_DATA_R = 0x01;               // rectangle output value 1
35.         timeelapse(10000);                          // pseudo wait
36.         GPIO_PORTK_DATA_R = 0x00;               // rectangle output value 0
37.         timeelapse(200);
38.     }
39. }
40.
41. //Keyboard
42. unsigned short evaluation(unsigned char *software_input)
43. {
44.     unsigned char k = 0;
45.     unsigned char countery = 0;
46.     unsigned short keyValue = 0xEFFF;
47.     static unsigned char counterx = 0;
48.
49.     if((*software_input & 0xF0) == 0x70)
50.     {
51.         counterx = 0;
52.     }
53.
54.     for(k = 0; k < COLUMNNUMBER; k++)
55.     {
56.         if((*software_input & 0xF) == 0xF)              // check input value of ports PK0 to PK3
57.         {
58.             continue;
59.         }
60.         else if((*software_input & (1 << k)) == 0)
61.         {
62.             switch(*software_input & 0xF0)
63.             {
64.             case 0x70:
65.                 if(k==0)
66.                         keyValue = 0x1;
67.                 else if(k==1)
68.                         keyValue = 0x4;
69.                 else if(k==2)
70.                         keyValue = 0x7;
71.                 else if(k==3)
72.                         keyValue = 0xA;
73.                 break;
74.             case 0xb0:
75.                 if(k==0)
76.                         keyValue = 0x2;
```

```
77.                 else if(k==1)
78.                         keyValue = 0x5;
79.                 else if(k==2)
80.                         keyValue = 0x8;
81.                 else if(k==3)
82.                         keyValue = 0x0;
83.                 break;
84.             case 0xd0:
85.                 if(k==0)
86.                         keyValue = 0x3;
87.                 else if(k==1)
88.                         keyValue = 0x6;
89.                 else if(k==2)
90.                         keyValue = 0x9;
91.                 else if(k==3)
92.                         keyValue = 0xB;
93.                 break;
94.             case 0xe0:
95.                 if(k==0)
96.                         keyValue = 0xF;
97.                 else if(k==1)
98.                         keyValue = 0xE;
99.                 else if(k==2)
100.                        keyValue = 0xD;
101.                else if(k==3)
102.                        keyValue = 0xC;
103.                break;
104.            }
105.
106.            countery++;
107.            counterx++;
108.        }
109.    }
110.
111.    if(1 < countery || 2 < counterx)//Check y double count and check x double count
112.        return 0xFFFF;
113.    else
114.        return keyValue;
115. }
116.
117. unsigned short keyboardscan(void)
118. {
119.    //0x7f = 0111 1111
120.    //0xbf = 1011 1111
121.    //0xdf = 1101 1111
122.    //0xef = 1110 1111
123.    unsigned char software_output[COLUMNNUMBER] = {0x7f, 0xbf, 0xdf, 0xef};
124.    unsigned char i = 0;
125.    unsigned short keyPress = 0xEFFF;
126.    for(i = 0; i < COLUMNNUMBER; i++)
127.    {
128.        GPIO_PORTK_DATA_R = (software_output[i] & 0xF0);                    // output of binary values
    x1,x2,x3,x4 into port K
129.
130.        timeelapse(578);                                                    // wait pseudo 100
131.
132.        software_output[i] = GPIO_PORTK_DATA_R;                    // read port K input value
133.        //set  incoming
134.        //oooo iiii
135.        unsigned short buf = evaluation(&(software_output[i])); //can be removed
136.        if(buf != 0xEFFF && keyPress != 0xFFFF)
137.        {
138.                keyPress = evaluation(&(software_output[i]));
139.        }
140.    }
141.    return keyPress;
142. }
143.
144. void keyboard()
145. {
146.    SYSCTL_RCGCGPIO_R |= (1<<9);                // clock enable port K
147.    while((SYSCTL_PRGPIO_R & (1<<9)) == 0);     // clock when port K is available
148.    GPIO_PORTK_DEN_R = 0xFF;                    // digital I/O pins enabled
149.    GPIO_PORTK_DIR_R = 0xF0;                    // define output direction I/O pins
150.
151.    while(1)
152.    {
153.        unsigned short scan = keyboardscan();
154.        if(0xFFFF == scan)
155.            printf("2 or more keys have been simultaneously pressed\n");
156.        else if(0xEFFF == scan)
```

```
157.         {
158.             continue;                                   // no key pressed
159.         }
160.         else
161.         {
162.                 char wert[] = {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
163.
164.             printf("key value: %c\n", wert[scan]);    // print key value in hex-code on console
165.             timeelapse(1605136);
166.         }
167.     }
168. }
169.
170. void main(void)
171. {
172.     printf("Start\n");
173.     //rectangleSignal();
174.     keyboard();
175. }
```

7 Literaturverzeichnis

[1] Versuchsbeschreibung Seite 4.

[2] Vorlesung “Peripherie-Register und GPIO-Ports” Folie 46, sowie Folie 54

[3] https://www.youtube.com/watch?v=yYnX5QodqQ4