# Laboratorium z przedmiotu Systemy wbudowane (SW)

## Karta projektu – zadanie 7

*Nazwa projektu:*

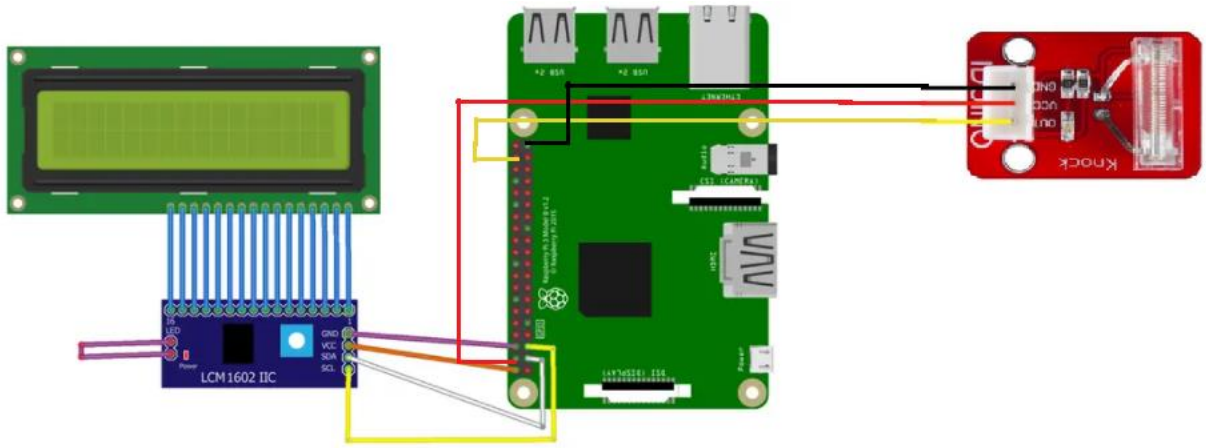
HIT-ME ALARM

| *Prowadzący:* mgr inż. Ariel Antonowicz | *Autorzy (tylko nr indeksu):*<br>144653 | *Grupa dziekańska:* | *I5.1* |
|---|---|---|---|
| | | ***Ocena:*** | |

*Cel projektu:*

Zbudowanie smart-budzika wyłączanego uderzeniem.

*Schemat:*

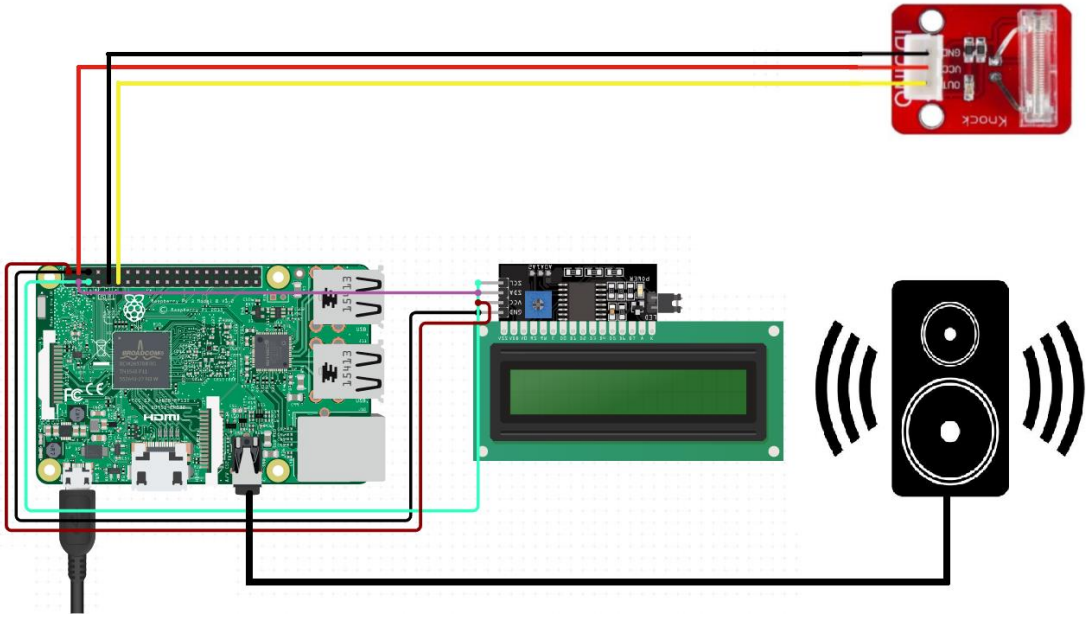



*Wykorzystana platforma sprzętowa, czujniki pomiarowe, elementy wykonawcze:*

*Raspberry Pi 4b 4GB*
*Iduino SE024 – czujnik uderzenia*
*LCD 2x16 I2C*
*Gumowa osłona na czujnik uderzenia*

1. Cel i zakres projektu
   Celem projektu było zbudowanie budzika wykorzystującego czujnik uderzeń do interakcji z użytkownikiem, który uderzony odpowiednią ilość razy wyłączy alarm grający z zewnętrznego głośnika. Do komunikacji oraz informowania użytkownika o alarmie miał służyć wyświetlacz 2x16 I2C.

2. Schemat połączeniowy



3. Projekt a realizacja (jakie były założenia, co się udało zrobić, jak można to rozwiązać)
   Projekt zakładał zbudowanie prostego budzika który wyświetla godzinę, następnie o danej godzinie wyświetla ile razy należy uderzyć czujnik aby wyłączyć muzykę. Projekt udało się rozbudować o dane elementy:
   - projekt nie wymaga wyświetlacza/program uruchamia się automatycznie
   - po uruchomieniu urządzenia ekran automatycznie wyświetla godzinę
   - po pojedynczym uderzeniu w czujnik uruchamia się program do ustawienia godziny alarmu
   - jeżeli użytkownik poda prawidłową godzinę (z zakresu 00:00 – 23:59) program informuje o ustawieniu budzika i dalej wyświetla godzinę
   - o odpowiedniej godzinie urządzenie zaczyna grać ustawioną muzykę oraz informuje użytkownika ile razy należy uderzyć czujnik w celu wyłączenia muzyki
   - program ostatecznie informuje o zakończeniu działania przez komunikat na wyświetlaczu

4. Najważniejsze fragmenty kodu

```
def mainLoop():
    alarmSet = False

    # Cleaning screen at beggining
    cleanDisplay()
    while alarmSet == False:
        channel = GPIO.wait_for_edge(17, GPIO.BOTH, timeout=1000)
        if channel is None:
            mylcd.lcd_display_string(time.strftime('%I:%M:%S %p'), 1)
            mylcd.lcd_display_string(time.strftime('%a %b %d, 20%y'), 2)
        else:
            cleanDisplay()
            # Getting hour and minute from user
            hour = setHour()
            minute = setMinutes()

            # Setting time
            alarmSet, alarmTime = checkHour(hour, minute)
            if alarmSet == True:
                schedule.every().day.at(alarmTime).do(job)

                # Printing allert
                cleanDisplay()
                mylcd.lcd_display_string(('Alarm set to:'), 1)
                mylcd.lcd_display_string(alarmTime, 2)
                time.sleep(1);
            cleanDisplay()


    while alarmSet == True:
        schedule.run_pending()
        time.sleep(1)
        mylcd.lcd_display_string(time.strftime('%I:%M:%S %p'), 1)
        mylcd.lcd_display_string(time.strftime('%a %b %d, 20%y'), 2)
```

Główna pętla programu, sprawdza czy użytkownik ustawił już alarm. W przypadku braku uderzeń program wyświetla godzinę. W przypadku wykrycia uderzenia w czujnik przechodzimy do trybu ustawiania godziny i minuty alarmu. Jeżeli godzina alarmu jest z prawidłowego zakresu, ustawiamy godzinę alarmu i przechodzimy do pętli z ustawionym alarmem. W tej pętli program nie reaguje na uderzenia a jedynie wyświetla czas oraz czeka na odpowiednią godzinę.

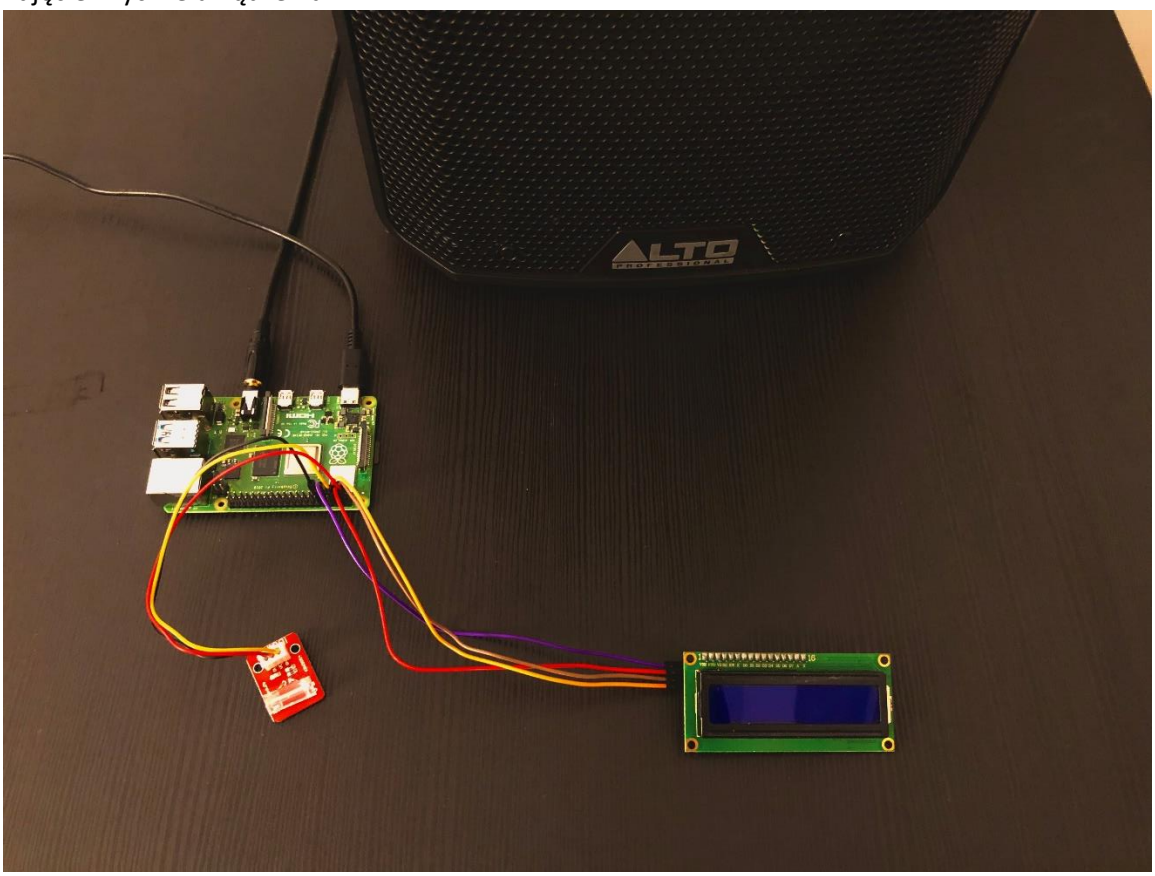
```
def setHour():
    timeForSetingHour = 30
    hour = 0

    cleanDisplay()
    mylcd.lcd_display_string('Set your hour:', 1)
    mylcd.lcd_display_string(str(hour), 2)

    while 0 < timeForSetingHour:
        channel = GPIO.wait_for_edge(17, GPIO.BOTH, timeout=1000)
        if channel is None:
            timeForSetingHour -= 1
        else:
            timeForSetingHour -= 1
            hour += 1
            mylcd.lcd_display_string('Set your hour:', 1)
            mylcd.lcd_display_string(str(hour), 2)

    return hour
```

Funkcja ustawiająca godzinę. Na identycznej zasadzie działa funkcja ustawiająca minuty. Użytkownik ma odpowiednio 30 sekund na ustawienie godziny i 60 sekund na ustawienie minut. Na wyświetlaczu cały czas wyświetlana jest ustawiona godzina, przez co użytkownik jest świadomy ile jego uderzeń już zostało zliczonych. W przypadku ustawiania godziny po 30 sekund program automatycznie przejdzie do ustawiania minut. W przypadku minut program automatycznie wróci do oryginalnej głównej pętli która następnie sprawdzi czy dane są prawidłowe.

```
def job():
    # Define how many times we should hit sensor to turn off the alarm
    hitTarget = random.randrange(5, 10)

    cleanDisplay()
    mylcd.lcd_display_string("Hit me:", 1)
    mylcd.lcd_display_string(str(hitTarget) + " times", 2)

    # Playing asynch sound
    playsound('kanye_alarm.wav', False)
    count = 0
    while count < hitTarget:
        channel = GPIO.wait_for_edge(17, GPIO.BOTH, timeout=1000)
        if channel is None:
            continue
        else:
            hitTarget -= 1
            mylcd.lcd_display_string(str(hitTarget) + " times", 2)

    cleanDisplay()
    mylcd.lcd_display_string("Thanks for" , 1)
    mylcd.lcd_display_string("using!", 2)
    sys.exit()
```

Główna funkcja alarmu, to ta funkcja opisuje co się stanie po dojściu do ustawionej godziny. Najpierw funkcja losuje liczbę z zakresu <5,10> i wyświetla ją na ekranie. Następnie z głośnika jest odtwarzana muzyka. Każde uderzenie czujnika powoduje zmniejszenie liczby wyświetlanej na ekranie. Gdy licznik dojdzie do 0 program wyświetla komunikat z podziękowaniem za używanie i wyłączy muzykę.

5. Zdjęcie fizyczne urządzenia



6. Link do prezentacji

https://youtu.be/W0HXmjOc-bs

7. Podsumowanie i wnioski

Cały projekt okazał się wyjątkowo przyjemny i pouczający. Był to mój pierwszy większy projekt na Raspberry PI. Jestem bardzo zadowolony z ostatecznego efektu gdyż udało mi się osiągnąć dokładnie to czego potrzebowałem, czyli awaryjny budzik który można uruchomić i ustawić bez włączania samego komputera (bez ekranu). Największym problemem okazało się zsynchronizowanie muzyki z programem, gdyż większość bibliotek nie umożliwia wyłączenia grania muzyki w trakcie trwania programu. Jedyna powszechnie znana biblioteka obsługująca asynchroniczne granie muzyki (playsound) przez większość internetu była uznawana jako nie współpracująca z ARM, jednak po doinstalowaniu odpowiednich pakietów udało się użyć tej biblioteki. Projekt można rozwinąć o odpowiednią obudową i przede wszystkim wprowadzenie zapętlonego uruchamiania (po wykonaniu alarmu program mógłby wracać do wyświetlania godziny). O ile pierwszy pomysł jest całkiem racjonalny i raczej czysto estetyczny, o tyle drugi może być już dosyć mocno niepraktyczny, ponieważ szkoda marnować tak drogie urządzenie jak Raspberry PI na zwykły zegar z alarmem.