Circuit Playground Express

Systèmes embarqués

Circuit Playground Express
Micro Python

Objectif :

Découvrir quelques aspects de la mise en œuvre de systèmes embarqués autour d'une carte peu onéreuse aux multiples possibilités la Playground Express d'Adafruit associée au langage micro python.

La carte est pilotée grâce à un microcontrôleur 32 bits ATMEL ATSAMD21, doté d'un cœur ARM® Cortex®-M0+ processor, opérant à une fréquence de 48 MHz.

SMART ARM-Based Microcontroller

(*Atmel est une branche de Microchip depuis son rachat en 2012 mais a conservé ses designs de circuits compacts et performants.*)

Quelques indications :

Une notion à retenir.

Attention requise danger pour le matériel.

Sommaire

1 TP_1 : Entrées détection de front

Circuit Playground Express
Micro Python

1.1 Objectif :

Prendre en compte les entrées dans une application nécessite quelques précautions. En effet face à la vitesse de traitement une action sur un bouton poussoir ne peut être prise en compte de manière sécurisée. Nous obtiendrons des détections multiples sans précautions particulières.

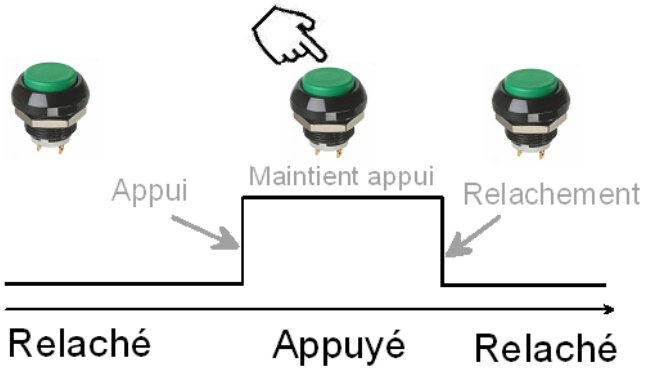
C'est l'objet de ce travail répondre à la question suivante : comment détecter des changements d'états des entrées pour aboutir à une maitrise de la prise en compte des informations.

1.2 Le bouton poussoir ou BP

Il peut avoir différentes dimensions, formes ….. Selon l'équipement ou le système …. Être équipé d'un voyant lumineux ou d'une led ……

Un bouton poussoir peut avoir quatre situations distinctes, deux états stables APPUYE ou REPOS et deux transitions d'un état stable vers un autre soit :

- **<u>L'appui</u>** ⇨ passage de l'état REPOS vers l'état APPUYE.
- **<u>Le relâchement</u>** ⇨ passage de l'état APPUYE vers l'état REPOS.

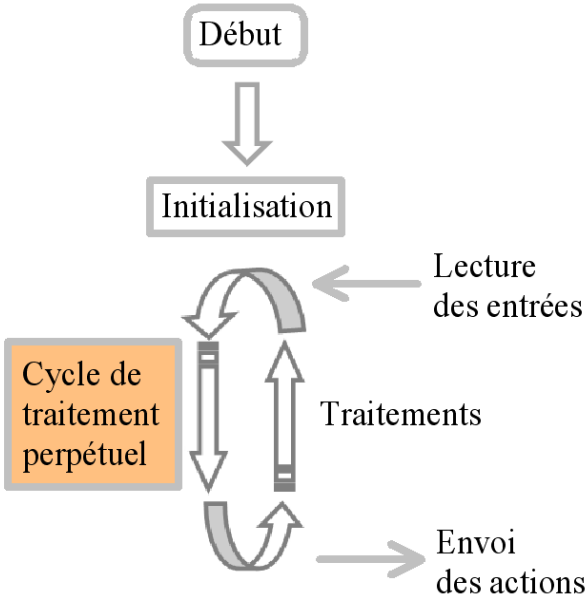


1.3 Fonctionnement d'un système embarqué

Un système embarqué est un système piloté par un processeur de manière autonome. Une fois démarré le fonctionnement du système se poursuit sans fin tant qu'il est alimenté.

Le cycle de fonctionnement est représenté sur le schéma ci-contre.

Après une initialisation lors de la mise sous tension de l'équipement celui-ci reproduit son cycle : lecture des entrées / traitements / envoi des sorties.



Q1. Donner quelques exemples de systèmes embarqués dans votre vie de tous les jours.

Q2. Le programme pilotant le système embarqué a-t-il une fin ?

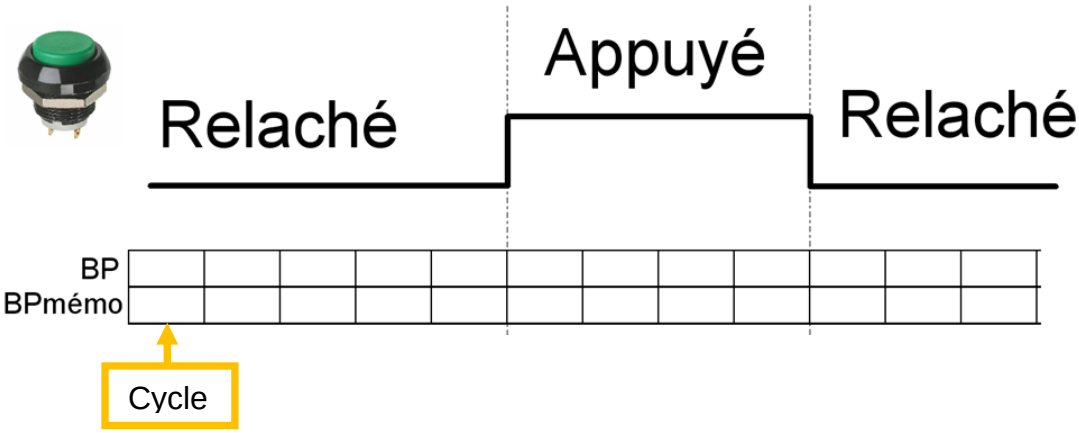
1.4 Détection des appuis

La détection des appuis est réalisée à chaque cycle du programme par une lecture de l'état des boutons poussoirs dans une variable BP suivie d'un traitement pour détecter les appuis ou relâchement des boutons. Une fois le traitement effectué l'état est mémorisé dans une variable BPmemo, cette variable contient donc l'état au précédent cycle lors de la détection des fronts.

Nous allons nous intéresser à l'algorithme permettant de détecter les fronts appui ou relâchement pour les boutons poussoirs. C'est-à-dire déterminer comment détecter un appui et comment détecter un relâchement.

Pour cela remplir le tableau de la page suivante ou chaque intervalle de temps représenté par un carré représente un cycle de fonctionnement de notre programme. Nous observons alors l'évolution des variables BP et BPmemo.

Q3. Compléter le diagramme ci-dessous. Réponses possibles A (appuyé) ou R (relaché repos)



Q4. Écrire la condition logique de détection d'un appui.

Q5. Écrire la condition logique de détection d'un relâchement.

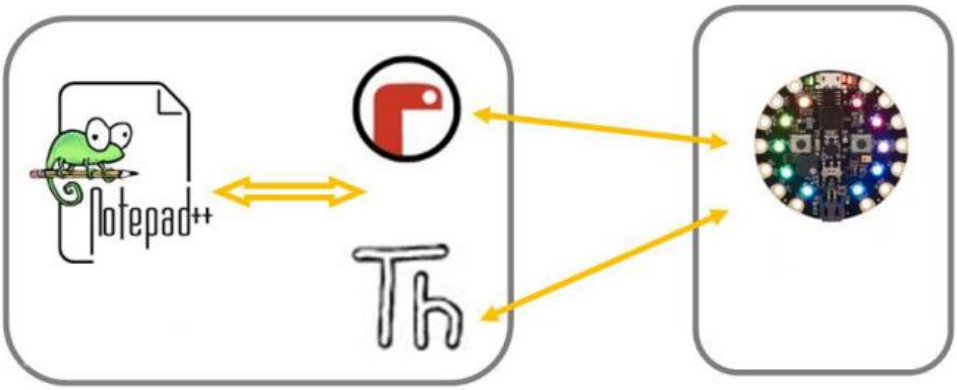
2 Mise en œuvre avec la carte Playground Express

2.1 Le programme à tester

Voilà le script microPython qui utilise la méthode décrite précédemment pour détecter et compter le nombre d'appui sur le bouton poussoir A : CPX_TP1_Detection et comptage appui BPA.py

```
# -*- coding: utf-8 -*-
#
# CPX_TP1_Detection et comptage appui BPA.py

import time
import board
import digitalio

# Initialisations matérielles
led = digitalio.DigitalInOut(board.D13)
led.switch_to_output()

buttonA = digitalio.DigitalInOut(board.BUTTON_A)
buttonA.switch_to_input(pull=digitalio.Pull.DOWN)

# Initialisations logicielles
memobuttonA = False
comptage_appui_A = 0

# Boucle perpétuelle
while True:

    if buttonA.value == True and memobuttonA == False:
        print ("BoutonA appuye")
        comptage_appui_A += 1
        print("Nombre d'appui BTA : ",comptage_appui_A)

    if buttonA.value == False and memobuttonA == True:
        print ("BoutonA relache")

    memobuttonA = buttonA.value

    time.sleep(0.01)
```

2.2 Expérimentations



Detection et comptage appui BPA BPB.py

code.py

```
Console
Nombre d'appui BTA :  6
BoutonA relache
BoutonA appuye
Nombre d'appui BTA :  7
BoutonA relache
BoutonA appuye
Nombre d'appui BTA :  8
BoutonA relache
```

Script1. Charger et tester le fonctionnement de la détection des appuis sur le bouton poussoir A

Script2. Modifier le script précédent pour ajouter la détection et le comptage des relachements du bouton poussoir B

3 Scripts avancés

Pour aller plus loin …..

3.1 Programmer un clignotement

Script3. Concevoir un script qui réalise dix clignotements de la led board.D13 pour chaque appui du switch A

3.2 Génération d'un code Morse

Le code morse a été inventé par Mr Samuel Morse en 1832 pour la transmission de message par télégraphie, ou signaux lumineux.

L'INVENTEUR :

Samuel Morse (1791-1872)

LE CODE MORSE :

Chaque lettre est codée avec une alternance de signaux courts ou longs:

- Signal long appelé trait
- Signal court appelé point

Chaque lettre est séparée par un espace.La durée d'un trait est égale à celle de trois points, un espace a une durée identique à celle du trait.

La durée d'un trait sera égale à 600 mS.

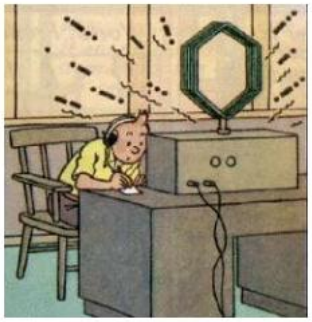
L'ALPHABET MORSE :

A	·–	N	–·	0	–––––
B	–···	O	–––	1	·––––
C	–·–·	P	·––·	2	··–––
D	–··	Q	––·–	3	···––
E	·	R	·–·	4	····–
F	··–·	S	···	5	·····
G	––·	T	–	6	–····
H	····	U	··–	7	––···
I	··	V	···–	8	–––··
J	·–––	W	·––	9	––––·
K	–·–	X	–··–	.	·–·–·–
L	·–··	Y	–·––	,	––··––
M	––	Z	––··	?	··––··

Script4. Réaliser un programme émettant le code morse du signal SOS : ··· ––– ··· avec la led.

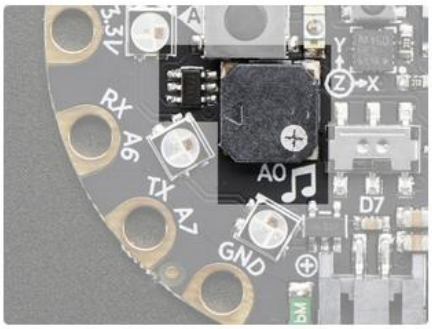
Script5. Réaliser un programme émettant le code morse du signal SOS en audio à partir des informations données sur la page suivante.

4 Et un vrai code Morse en audio



Un code Morse envoyé avec un signal lumineux c'est bien mais envoyé en audio c'est encore mieux.

4.1 La sortie audio de la carte CPX



The speaker is over here, its small but can make some loud sounds! You can **ENABLE** or disable the speaker. If you disable the speaker, audio will only come out the **A0/AUDIO** pin. If you enable the speaker, audio will come out from both!

https://learn.adafruit.com/adafruit-circuit-playground-express/circuitpython-audio-out

4.2 La génération de fréquences simples

Pour générer des sinusoïdes de son il suffit de créer mathématiquement une période complète de la sinusoïde désirée. Ensuite on peut la 'jouer' sur la sortie audio après avoir validé et configuré celle-ci.

Création de la sinusoïde

```
import array
import math
from audiocore import RawSample
from audioio import AudioOut

# Génération d'une période de la sinusoïde
# 440 Hz middle 'A'
FREQUENCY = 440
# 8000 échantillons / secondes
SAMPLERATE = 8000
# Calcul des valeurs placées dans l'array sine_wave
length = SAMPLERATE // FREQUENCY
sine_wave = array.array("H", [0] * length)
for i in range(length):
    sine_wave[i] = int(math.sin(math.pi * 2 * i / length) \
                       * (2 ** 15) + 2 ** 15)
```

Validation du haut-parleur de la carte

```
# Validation du haut-parleur de la carte
speaker_enable = digitalio.DigitalInOut(board.SPEAKER_ENABLE)
speaker_enable.direction = digitalio.Direction.OUTPUT
speaker_enable.value = True
audio = AudioOut(board.SPEAKER)
# Préparation du signal
sine_wave_sample = RawSample(sine_wave)
```

Tout est prêt : envoi du signal

```
# Envoi du son sur la sortie speaker
audio.play(sine_wave_sample, loop=True)
time.sleep(5)
audio.stop()
```

Le code ici : CPX_TP1_Premier Son.py