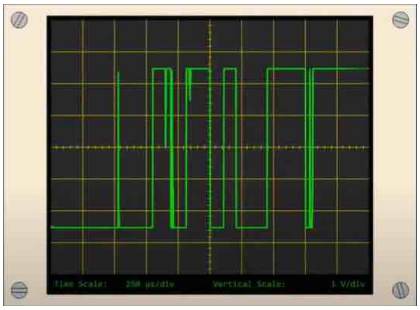
Druk op de knop 4

tekst en schema's **Peter Knijff**

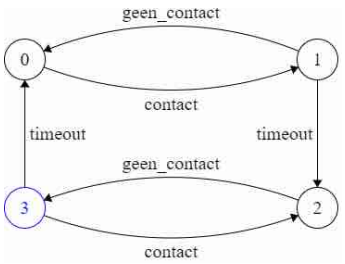
In de vorige aflevering van deze serie “Pre-Emptive State Machine” is het geluid in combinatie met een AKI behandeld. Het zal natuurlijk mooier zijn wanneer dit uitgebreid kan worden met bewegende slagbomen; oftewel een AHOB.

Omdat de ledjes en het geluid van de AKI onafhankelijk van elkaar kunnen werken, zal het vrij eenvoudig zijn om de slagbomen met een servo te laten bewegen; zeker als deze ook pre-emptive te maken is.

Het maken van een betrouwbare button



Wanneer een aan/uit knop gebruikt wordt dan zal het contact niet altijd mooi aan of uit zijn. Door mechanische beperkingen, zoals de veerkracht van het lipje, zal de knop af en toe het contact maken of onderbreken. Dit is goed te zien op een oscilloscoop:



De button kan betrouwbaarder worden. Dit kan verbeterd worden door, wanneer de button in- of uitgedrukt wordt, een bepaalde tijd te wachten totdat dit stabiel is. Een normale wachttijd is ongeveer 50-100 ms maar kan natuurlijk aangepast worden.

Ook hier kan een statemachine voor gemaakt worden. Wanneer de status 0 of 1 is, dan zal de statemachine de statuswaarde LOW hebben. In status 2 of 3 zal dit LOW zijn.

De tabel zal zijn:

State	initialisatie	event	volgende status	Wat doet deze state
0	Geen initialisatie nodig	Een contact is gedetecteerd	1	Wachten tot een contact gedetecteerd is
1	Zet timer wit aan	Contact is uit	0	
		Timeout (100 ms)	2 en zet status HIGH	Wacht tot contact blijvend is
2	Geen initialisatie nodig	Contact is los	3	Wachten tot contact los is
3	Ze timer aan	Contact is weer aan	2	
		Timeout (100 ms)	0	Wacht tot contact blijvend los is

Hier is natuurlijk ook een preemptive class van gemaakt: CDebounce. Aan het begin van de loop() functie wordt de CDebounce object gebruikt:

```
void loop() {
  unsigned long currentMillis = millis();
  int iButtonState = cDebounce.go(currentMillis);
```

Bron: Het Zijspoor - clubblad van Modelbouwvereniging Arnhem e.o.

De variabele iButtonState (de status van de button) wordt dus niet meer bepaald door de functie “digitalRead()” maar door de state van het debounce object; de wordt geretourneerd door de functie go(). De volledige beschrijving van de CDebounce class staat beschreven in het document “TipsEnTricks.docx” temeer omdat dit een heel eenvoudige class is welke veel debounce problemen kan voorkomen.

Hoe werkt een AHOB

Hoe een AHOB werkt is hier te zien: https://www.youtube.com/watch?v=8x7iMM6tfew&t=44s. Deze schakeling wordt gemaakt aan de hand van dit filmpje.

Tijd (s)	Wat gebeurt er?
0:01	De bel gaat en de lampen knipperen.
0:06	De slagbomen gaan sluiten
0:18	De slagbomen zijn gesloten
0:28	De trein is voorbijgegaan
0:31	De bel gaat uit en de slagbomen gaan open
0:43	De bomen zijn open en de lichten gaan uit

Dit zijn de tijden die op het filmpje te zien zijn maar op een modelbouwbaan kunnen de tijden natuurlijk aangepast worden. In deze aflevering zal dus een pre-emptive servobesturing gemaakt worden waarna de servo toegevoegd wordt aan de schakeling.

Het maken van een servo schakeling

BELANGRIJK! Natuurlijk is het hart van de schakeling een servo, maar er moet opgelet worden welke servo aangeschaft wordt. We hebben een 180° servo nodig; deze kan 180 graden draaien. Dit is de meest gangbare servo. Echter, er is een tweede type servomotor: de zgn. “Continuous Rotation Servo” of CRS. Deze draait continue afhankelijk van de stroom. Ze kunnen soms (bijna) hetzelfde uitzien zoals de foto laat zien. Beide gekocht bij Ali maar de linker is de 180° servo terwijl de rechter de CRS is.

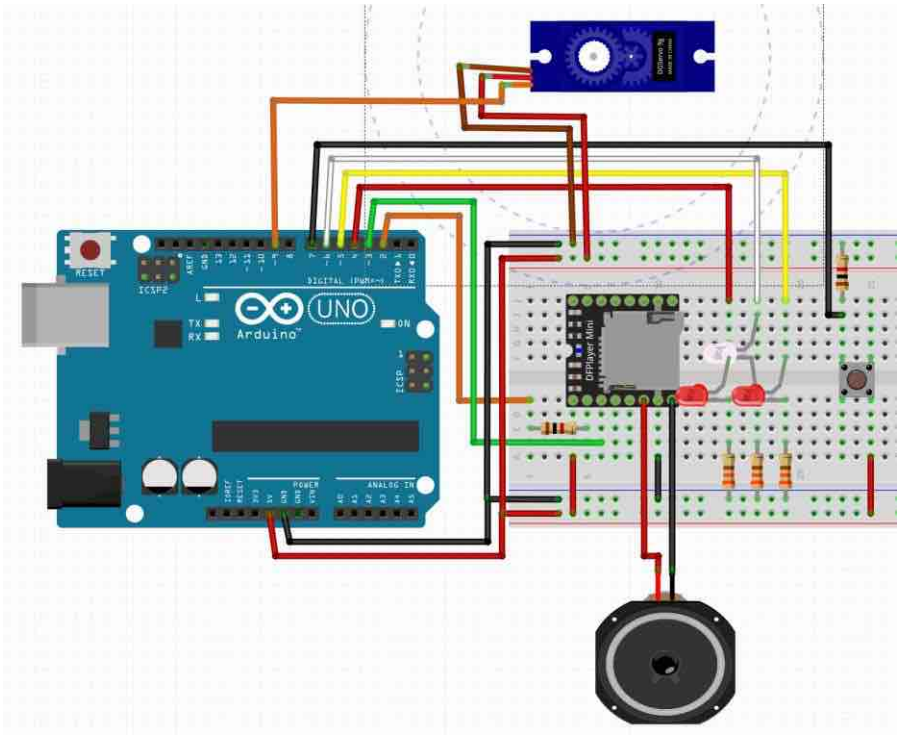
Aanstuurgegevens voor de servo

De servo-aansluitingen zijn zeer eenvoudig:

Bruin of zwart	GND
Rood	5V (voeding)
Geel/Oranje	PWM signaal

PWM signaal (Puls-Width Modulation, pulsbreedtemodulatie) zijn bij een Arduino UNO of nano de pinnen 3, 5, 6, 9, 10 en 11. In deze aflevering gebruiken we pin 9 als het PWM signaal. De aansluitingen op de Arduino zijn weergegeven in het Fritzing bestand AHOB_Geluid.fzz:

- Er is dus weer een button gedefinieerd welke is aangesloten op pin 7
- De witte led is aangesloten op pin 6
- De rode leds zijn aangesloten op pin 5 resp. pin 4
- Om de Dfplayer aan te sturen worden de pinnen 3 en 2 gebruikt
- En pin 9 (een PWM pin) wordt gebruikt om de servo aan te sturen.



Om de servo aan te kunnen sturen zijn een paar berekeningen nodig. De servo kan aangestuurd worden met een PWM waarde van 0 t/m 255. 255 is dan 180°. De waarde van de PWM pin voor de hoek is dus: waarde = hoek * 255 / 180.

De servo zal in een aantal stappen (n) van de minimale naar de maximale uitslag moeten gaan. Wanneer de totale tijd (T) bepaald is dan is een stapje voor de servo: t_{stap} = T / n. Bij de initialisatie van de servo zijn dus de volgende gegevens nodig:

1. De PWM signaal pin
2. Het aantal stappen dat de servo moet maken
3. De tijd dat de slagbomen van gesloten naar geopend gaan (en omgekeerd)
4. De starthoek waar de servo begint (hoeft niet altijd 0° te zijn)
5. De eindhoek waar de servo begint (hoeft niet altijd 180° te zijn)

De code voor de servo pre-emptive state machine

De initialisatie van de servo is:

```
init(int iPinNr, long lNrSteps, long lTime, long lStartHoek, long lEind-
Hoek)
```

Stel de slagboom gaat, wanneer deze open gaan, van 20° naar 130° en dat dit gebeurt in 4 secondes. We wensen 110 stappen (1° per stap) dan zal de initialisatie van de servo zijn:

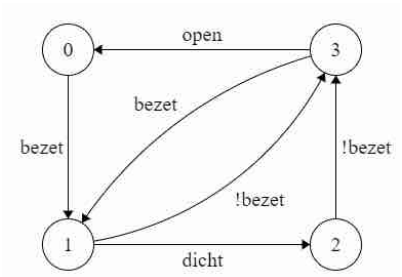
```
Init(9, 110, 4, 20, 30);
```

Daarnaast is de go functie dezelfde als voor eerder beschreven pre-emptive state machines:

```
go(currentMillis, bBezet);
```

De statemachine voor de servo

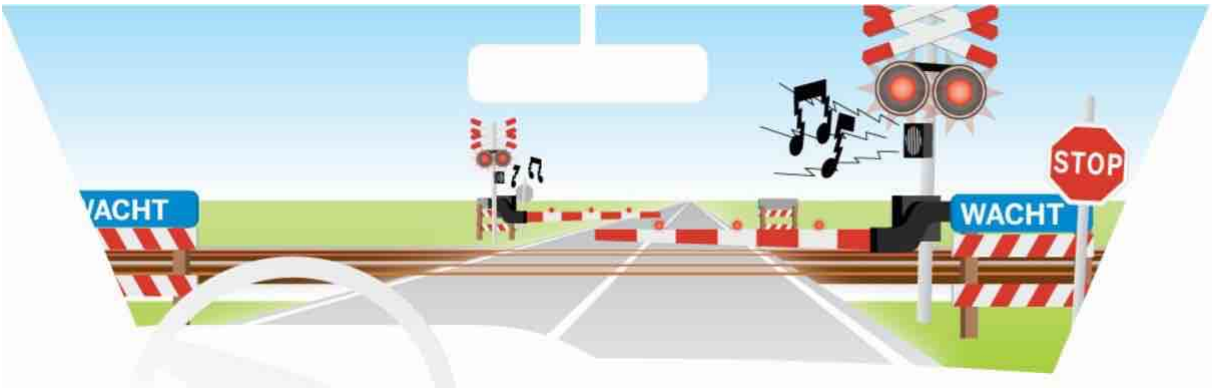
Zoals normaal, voordat de software beschreven wordt, wordt eerst de statemachine gemaakt. Wanneer de bomen open zijn en het baanvlak wordt bezet dan moeten de bomen naar



beneden gaan en vice versa. Maar wanneer bijvoorbeeld de bomen opengaan en het baanvlak raakt weer bezet dan moeten de bomen direct dichtgaan. Hetzelfde geldt wanneer de bomen dichtgaan en de trein vertrekt uit het baanvlak.

De bijbehorende statemachine tabel is dan:

state	initialisatie	event	volgende status	wat doet deze state
0	starthoek	baanvak bezet	state 1	de servo gaat dicht
1	servo gaat dicht	servo is dicht	state 2	de servo stopt
		baanvak onbezet	state 3	de servo gaat open
2	eindhoek	baanvak onbezet	state 3	de servo gaat open
3	servo gaat open	servo is open	state 0	de servo stopt
		baanvak bezet	state 1	de servo gaat dicht



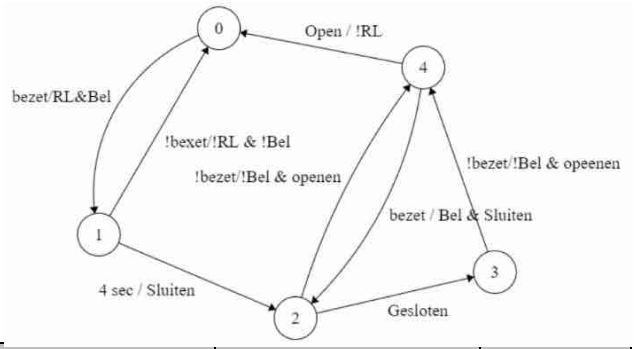
De uiteindelijke schakeling

Ook hier eerst een statemachine:

Bel = bel aan; !Bel = bel uit
RL = rode leds aan; !RL = rode leds uit

bezet = treinstuk bezet
!bezet = treinstuk vrij

De bijbehorende statemachine tabel is dan:



state	initialisatie	wat doet deze state	event	volgende status
0	Rode leds uit, bel is uit en bomen zijn open	Dit is de rusttoestand van de AHOB	Baanvlak is bezet	state 1
1	Rode leds knipperen en bel gaat aan	Wachten totdat de bomen dicht gaan	4 secondes voorbij	state 2
			Baanvlak is leeg	state 0
2	Bomen gaan dicht en bel gaat aan	Bomen bewegen totdat deze dicht zijn	Bomen zijn dicht	state 3
			Baanvlak is leeg	state 4
3	-	Wacht tot het baanvlak leeg is	Baanvlak is leeg	state 4
4	Bomen gaan open Bel gaat uit	Deze state wacht tot de trein weg is	Baanvlak weer bezet	state 2
			Bomen zijn open	state 0

In deze statemachine worden de bomen natuurlijk door de servo bediend. De uiteindelijk code voor deze schakeling, inclusief de CServo code is te vinden in het document “04_PreemptiveState_add.docx”