Druk op de knop 2

Tekst en schema's **Peter Knijff**; tekstbewerking **Hans van de Ven**

Vooraf: in het vorige Zijspoor vertelde ik te zijn uitgedaagd door de vraag van de tweerail analoge baan naar 'geluiden voor het station'. Ik betoogde toen dat geluiden nooit alléén komen, maar altijd in combinatie met licht, beweging, et cetera. Ook het synchroon knipperen van allerlei zwaailichten van de hulpdiensten bij een uitgebeeld ongeval triggerden me en zo liet ik zien dat een simpele minicomputer als de Arduino, mits goed geprogrammeerd, hier veel in kan betekenen. Ik introduceerde toen het fenomeen 'Pre-Emptive State Machine', een specifieke programmeercode die er voor zorgt dat zaken in de gewenste volgorde en tijd worden afgewerkt.

Zo wordt er alleen actie ondernomen wanneer aan een bepaalde conditie voldaan is, bijvoorbeeld een time-out(), en hierdoor lijkt het alsof de Arduino meerdere taken tegelijkertijd kan doen. Dit is uitgelegd aan de hand van een knipperlichtinstallatie waarbij twee ledjes onafhankelijk van elkaar gingen knipperen.

Voor sommige leden is dit allemaal zeer interessant, voor anderen niet onmiddellijk bruikbare kost. Ook voor later staat daarom alle benodigde informatie (de afleveringen voor Het Zijspoor, maar ook artikelen die dieper op de materie ingaan) in mijn eigen cloudruimte. Kijk maar eens in de Github repository https://github.com/knijff1961/MVA. Daar zijn, naast dit document 02_PreemptiveState.docx, voor deze aflevering ook twee andere documenten toegevoegd:

- 02_PreemptiveState_add.docx, met extra informatie over de code.
- TipsEnTricks.docx, met informatie over hoe om te gaan met de Arduino en Arduino IDE

Zoals bij alle ICT-projecten zullen ook hier regelmatig verbeteringen doorgevoerd worden. Helaas gebeurde dit ook met de libraries (bibliotheken) waarin de class CPreEmptiveTimer beschreven is. Deze zullen dus opnieuw vanuit GitHub geïnstalleerd moeten worden. (zie TipsEnTricks.docx)

In deze tweede aflevering worden drie belangrijke onderdelen beschreven welke het maken van pre-emptive state machines vereenvoudigen.

- Gebruik maken van #define’s
- Grafisch beschrijven van een pre-emptive state machine
- Verbeteren van de switch-statements

Daarnaast wordt code behandeld van een AKI (Automatische Knipperlicht Installatie) beschreven welke gebruikt kan worden op elke (Nederlandse) modelbaan.

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

Gebruik maken van #define's

#defines worden gebruikt om de code niet alleen leesbaarder te maken maar ook makkelijker te onthouden. Zo kunnen de pinnen van de Arduino in plaats van een nummer ook een naam krijgen. Overal waar deze naam in de code gebruikt wordt, zal dit nummer door de IDE ingevuld worden.

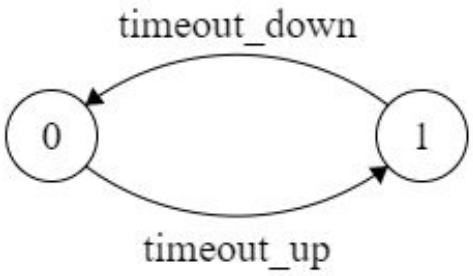
Hetzelfde geldt dus ook voor de timeout's, zowel voor de up (inschakelen) alsook voor de downs (uitschakelen).

► In 02_PreemptiveStateCode_add.docx hoofdstuk 2 wordt uitgebreid uitgelegd hoe de define's in de code gebruikt kunnen worden.

De code is daardoor opeens een stuk leesbaarder geworden en de pinnen respectievelijk de tijden zijn eenvoudiger aan te passen. Ook is het simpeler om de tijden dat een led aan of uit staat te veranderen. Hiermee kan eenvoudig geëxperimenteerd worden.

Grafisch beschrijven van een pre-emptive state machine

De knipperlichtinstallatie uit aflevering 1 is zeer eenvoudig; het heeft maar twee states: aan of uit. Vaak is het handig om een pre-emptive state machine grafisch weer te geven. Vooral wanneer er meerdere states beschreven moeten worden. Een mogelijke beschrijving van het knipperlicht is dan:



De state begint bij state 0 (led is aan). Na verloop van tijd zal de timeout_up optreden welke aangeeft dat de tijd dat de led aanstaat voorbij is. Nu moet er overgegaan worden naar de volgende state: state 1. Dit wordt eenvoudig aangegeven door een pijl met de event "timeout_up". Wanneer de timeout_down event optreedt, dan gaat de pre-emptive state machine terug naar state 0 en de led gaat aan. In een tabel ziet het er als volgt uit:

state	initialisatie	event	volgende status	Wat doet deze state
0	Zet led aan en zet timer up	timeout_up	1	Led is aan
1	Zet led uit en zet timer down	timeout_down	0	Led is uit

Aan de hand van het plaatje en de tabel kan nu eenvoudig elke state geprogrammeerd worden. Later krijgen we te zien dat één of meerdere events de state machine naar een andere state kunnen brengen.

NB: er zijn veel applicaties om statemachines te tekenen. Voor dit plaatje werd gebruikt:
https://www.cs.unc.edu/~otternes/comp455/fsm_designer/

Verbeteren van de switch-statements

De state machine werkt naar behoren. Er kunnen echter nog drie zaken verbeterd worden:

1. De code van de statemachine begint met if(cPreEmptiveTimer->timeout(currentMillis))
2. Daarna volgt de "cPreEmptiveTimer1.iState" switch en indien er een timeout is dan wordt er pas bekeken welke state de machine heeft: 0 of 1.
3. De initialisatie van de volgende state wordt in de vorige state bepaald.

Punt 1 en 2 zijn eenvoudig op te lossen door de statements te verwisselen. In pseudo code ziet het er dan als volgt uit:

```
IF STATE = 0 THEN
  IF EVENT is opgetreden
    DO acties
IF STATE = 1 THEN
  IF EVENT is opgetreden
    DO acties
Etc…
```

► Deze aanpassing is volledig beschreven in 02_PreemptiveStateCode_add.docx hoofdstuk 3.1

In het derde geval zou het beter zijn wanneer de initialisatie gedaan wordt daar waar het behoort: in de nieuwe state. Voor punt 3 worden twee nieuwe sub-states geïntroduceerd: de STATE_INIT en de STATE_WAIT

De STATE_INIT initialiseert de state terwijl de STATE_WAIT wacht tot één of meerdere events zijn opgetreden. (De sub-states STATE_INIT en STATE_WAIT zijn #defines welke gedefinieerd zijn in de CPreEmptiveTimer library.)

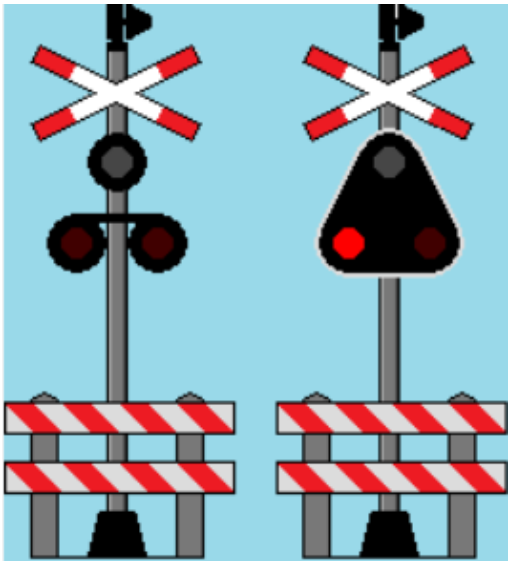
De code wordt nu iets uitgebreider: In elke state staan dus deze sub-states:

```
IF STATE = X THEN
  IF STATE_INIT THEN
    Initialiseer acties
  ELSE
    IF EVENT is opgetreden
      Spring naar state Y

IF STATE = Y THEN
Etc…
```

► Deze aanpassing is volledig beschreven in 02_PreemptiveStateCode_add.docx hoofdstuk 3.2

De AKI: Automatische Knipperlicht Installatie

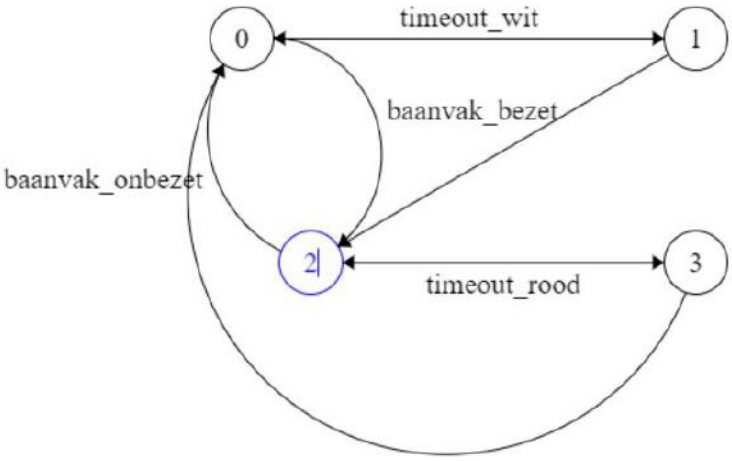


Iedereen kent wel de AKI; hoewel vele overwegen vervangen worden door AHOB's kunnen ze nog steeds gevonden worden. Het bekende wit knipperende licht wanneer het veilig is en de sneller knipperende rode lampen wanneer er een trein aan komt.

Wat we nu weten is dat er twee belangrijke events zijn: baan is veilig/onveilig en het knipperen, zowel voor het witte licht alsook de rode lichten. Het knipperen wordt gedaan door één CPreEmptiveTimer object.

Wanneer we een state diagram maken kan deze er als volgt uitzien:

Een state kan veranderen door meerdere events. Soms moet bij een event tijdens het overgaan naar de volgende status nog iets afgehandeld worden, bijvoorbeeld de witte led moet uit gaan wanneer er een trein in het baanvlak komt.



De bijbehorende statemachine tabel is dan:

State	initialisatie	event	volgende status	Wat doet deze state
0	Zet witte led aan Zet timer wit aan	baanvak_bezet	Zet witte led uit 2	Witte led aan
		timeout_wit	1	
1	Zet witte led uit Zet timer wit aan	baanvak_bezet	2	Witte led uit
		timeout_wit	0	
2	Zet rood links aan Zet rood rechts uit Zet timer rood aan	baanvak_onbezet	Zet rode led links uit 0	Rode led links aan Rode led rechts uit
		timeout_rood	3	
3	Zet rood links uit Zet rood rechts aan Zet timer rood aan	baanvak_onbezet	Zet rode led rechts uit 0	Rode led links uit Rode led rechts aan
		timeout_rood	2	

In het begin lijkt dit vrij lastig, maar wanneer er met de vinger over de grafiek gegaan wordt en dan in de tabel gelezen wordt wat er moet/gaat gebeuren, dan valt dit wel mee.

► In 02_PreemptiveStateCode_add.docx hoofdstuk 4 wordt de code van de AKI verder behandeld.

In de volgende sessie gaan we de AKI uitbreiden, namelijk met een geluid-chip welke ook eenvoudig aangestuurd kan worden met de Arduino. Ook hebben we aandacht voor de de-bounce-knop. Welke knop? Dat gaan we uitleggen in deel 3.

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