Vooraf:

In de eerste aflevering van deze serie “Pre-Emptive State Machine” is de basis van een pre-emptive statemachine uitgelegd Zo wordt er alleen actie ondernomen wanneer aan een bepaalde conditie voldaan is, bijvoorbeeld een timeout(), en hierdoor lijkt het alsof de Arduino meerdere taken tegelijkertijd kan doen. Dit is uitgelegd aan de hand van een knipperlicht installatie waarbij twee knipperlichten onafhankelijk van elkaar gingen knipperen.

In de Github repository (<https://github.com/knijff1961/MVA>) zijn, naast dit document, 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 zal dus opnieuw vanuit GitHub geïnstalleerd moeten worden. (zie TipsEnTricks.docx)

In dit document worden drie belangrijke onderdelen beschreven welke het maken van preemptive state machines vereenvoudigen.

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

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

**Gebruik maken van #define’s**

Afbeelding met tekst, Lettertype, schermopname, Graphics

Automatisch gegenereerde beschrijving#defines worden gebruikt om de code niet leesbarder te maken maar ook makkelijker te onthouden. Zo kunnen de pinnen van de Arduino i.p.v. 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 alsook voor de downs.

In 02\_PreemptiveStateCode\_add.docx hoofdstuk 2 wordt uitgebreid uitgelegd hoe de define’s in de code gebruikt kan worden.

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

**Grafisch beschrijven van een preemptive state machine**

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

Afbeelding met schets, tekening, diagram, cirkel

Automatisch gegenereerde beschrijvingDe 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 optreed, dan gaat de preemtive 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 kan 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 echter er kunnen 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 in welke state de machine heeft: 0 of 1.
3. De initialisatie van de volgende state wordt in de vorige state bepaald.

De eerste twee punten kunnen in één oogopslag aangepast worden; draai ze om. Eerst de iState switches en in elke iState switch de event(s).

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 behoord: in de nieuwe state.

Voor punt 3 worden 2 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 een 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

Afbeelding met Verkeersbord, verkeerslicht

Automatisch gegenereerde beschrijving**De AKI: Automatisch Knipper 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 rode lampen knipperen wanneer er een trein aankomt.

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.

Afbeelding met cirkel, diagram, lijn, tekst

Automatisch gegenereerde beschrijvingWanneer 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 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 | ~~Zet witte leds uit~~  2 | Witte led uit |
|  |  | timeout\_wit | 0 |  |
| 2 | Zet rood links aan  Zet rood rechts uit | baanvak\_onbezet | Zet rode links led uit  0 | Rode led links aan  Rode led rechts uit |
|  | Zet timer rood aan | timeout\_rood | 3 |  |
| 3 | Zet rood links uit  Zet rood rechts aan | baanvak\_onbezet | Zet rode rechts led uit  0 | Rode led links uit  Rode led rechts aan |
|  | Zet timer rood 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

De volgende sessie gaan we de AKI uitbreiden namelijk met een geluid-chip welke ook eenvoudig aangestuurd kan worden met de Arduino.