Vooraf:

In de eerste aflevering van deze serie “Pre-Emptive State Machine” is de basis van een pre-emptive statemachine uitgelegd:

* Er wordt alleen actie ondernomen wanneer aan een bepaalde conditie voldaan is, bijvoorbeeld een timeout()
* Hierdoor lijkt het alsof de Arduino meerdere taken tegelijkertijd kan doen.
* Bovenstaande 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**

De vorige keer hebben we twee eenvoudige knipperlichten gemaakt die onafhankelijk met verschillende snelheden knipperde. De code werkt goed maar was niet zo mooi geschreven. Wanneer bijvoorbeeld een andere pin gebruikt gaat worden (bijvoorbeeld pin 11 moet pin 12 worden), dan moet overal in de code waar pin 11 gebruikt wordt, dit veranderd worden in 12. Dit is vrij intensief werk maar ook foutgevoelig; wanneer dit proces op één plek vergeten wordt, dan werkt het programma niet meer of niet meer goed. Een oplossing is dat de #define statement in de code gebruikt wordt. Met de #define wordt door de compiler een woord vervangen door een waarde. Zo kan men pin 11 definiëren als zijnde de eerste led. Een mogelijke #define zal dan kunnen zijn:

#define PIN\_EERSTE\_LED 11

Net als bij een variabele geldt ook hier: de naam begint met een letter of een underscore, gevolgd door één of meerdere letters, cijfers of underscores. Alleen bij #defines worden meestal hoofdletters gebruikt maar het is niet verplicht.

Hetzelfde kan dan ook gedaan worden met de timeout’s van de knippersnelheid. In de code gebruikte ik de naam:

* LEDx\_STATE\_UP De tijd dat led x aan staat
* LEDx\_STATE\_DOWN De tijd dat led x uit staat

(x is de nummer van de led)

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.

Deze geüpdatet code staat in de file CPreEmptiveTimerSimple\_v2.ino. Verdere verbeteringen zijn geïmplanteerd in CPreEmptiveTimerSimple\_v3.ino en CPreEmptiveTimerSimple\_v4.ino in het document 02\_PreemptiveStateCode\_add.docx hoofdstuk 2

**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  Zet timer up | timeout\_up | 1 | Led is aan |
| 1 | Zet led uit  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.

**Verbeteren van de switch-statements**

De state machine werkt naar behoren echter er kunnen nog twee 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.

In het eerste geval zal in de if statement met “cPreEmptiveTimer1.iState” maar één event tegelijkertijd bekeken worden namelijk de timer() moet afgelopen zijn. Dit is niet altijd wenselijk. Dit zal helemaal duidelijk worden bij de AKI waarbij gekeken moet worden of er een trein aanwezig is in een blok; er is dus een tweede event welke afgehandeld moet worden: trein komt in/verlaat het blok.

In het tweede geval zou het eenvoudiger zijn wanneer eerst bepaald wordt in welke state de machine is om vervolgens pas te kijken of er binnen de state een event heeft plaatsgevonden.

Voor het derde geval zou het beter zijn wanneer de initialisatie gedaan wordt daar waar het behoord: in de nieuwe state.

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 = 1 THEN

IF EVENT is opgetreden

DO acties

IF STATE = 1 THEN

IF EVENT is opgetreden

DO acties

Deze aanpassing is volledig beschreven in 02\_PreemptiveStateCode\_add.docx hoofdstuk 3.1

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:

IF STATE = 1 THEN

IF STATE\_INIT THEN

Initialiseer acties

ELSE

IF EVENT is opgetreden

Spring naar state 1

IF STATE = 1 THEN

IF STATE\_INIT THEN

Initialiseer acties

ELSE

IF EVENT is opgetreden

Spring terug naar state 0

Deze aanpassing is volledig beschreven in 02\_PreemptiveStateCode\_add.docx hoofdstuk 3.2

**De AKI: Automatisch Knipper Installatie**

Iedereen kent wel de AKI; hoewel de overwegen vervangen worden door AHOB’s kunnen ze nog steeds gevonden worden. Het bekende wit knipperende licht wanneer het veilig is en rooden lampen knipperen wanneer er een trein aankomt.

Afbeelding met lijn, tekst, diagram, cirkel

Automatisch gegenereerde beschrijvingWat we nu weten is dat er twee belangrijke events zijn: baan is veilig/onveilig en het knipperen. Het knipperen wordt weer 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 witte led moet uit gaan wanneer er een trein in het baanvlak komt.

De bijbehorende 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 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\_bezet | Zet witte led uit  0 | Rode led links uit  Rode led rechts aan |
|  | Zet timer rood aan | timeout\_wit | 2 |  |

In het begin lijkt dit vrij lastig maar wanner 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