Gedetailleerde informatie

Voor de serie “Pre-Emptive State Machine” deel 2

**Noot:** kan gedownload worden van:

https://github.com/knijff1961/MVA

Auteurs:

Datum : 04-08-2023

Versie : 0.9

Status : Review

Inhoud

[Content 2](#_Toc142038671)

[Afkortingen 3](#_Toc142038672)

[1 Introductie 4](#_Toc142038673)

Afkortingen

|  |  |
| --- | --- |
| MVA | Modelbouw vereniging Arnhem en omstreken |
| IDE | Integrated Development Environment; IDE is software die een ontwikkelaar helpt bij het ontwikkelen van software |
| Sketch | De naam die Arduino gebruikt voor een programma. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Introductie

Dit document beschrijft extra informatie over deel twee van de serie “Pre-Emptive State Machine”.

In dit document wordt gedetailleerder ingegaan op de beschrijving die in Het Zijspoor beschreven is. (01\_PreemptiveState.docx”}

# De #define (CPreEmptiveTimerSimple\_v2.ino)

In plaats van elke keer eenzelfde tekst te schrijven kan gebruik gemaakt worden van de #define statement. Voor de CPreEmptiveTimerSimple\_v2 regels 3-6 hebben alle pin waardes een #define gekregen:

#define PIN\_EERSTE\_LED 11

#define PIN\_TWEEDE\_LED 10

#define PIN\_DERDE\_LED 6

#define PIN\_VIERDE\_LED 5

Daarnaast zijn ook de tijden (in ms) voor elke led opgegeven voor de aan status (LEDx\_STATE\_UP) alsook de uit status (LEDx\_STATE\_DOWN). Deze zijn beschreven op regel 8-15:

#define LED1\_STATE\_UP 1000 // 1 sec

#define LED1\_STATE\_DOWN 1000 // 1 sec

#define LED2\_STATE\_UP 700 // 700 ms

#define LED2\_STATE\_DOWN 700 // 700 ms

#define LED3\_STATE\_UP 650 // 650 ms

#define LED3\_STATE\_DOWN 650 // 650 ms

#define LED4\_STATE\_UP 1200 // 1200 ms

#define LED4\_STATE\_DOWN 1200 // 1200 ms

Nu kunnen de getallen uit de code gehaald worden, beginnende met de setup() routine. De setup (regel 22 t/m 36) is dan als volgt:

void setup() {

pinMode(PIN\_EERSTE\_LED, OUTPUT);

pinMode(PIN\_TWEEDE\_LED, OUTPUT);

pinMode(PIN\_DERDE\_LED, OUTPUT);

pinMode(PIN\_VIERDE\_LED, OUTPUT);

cPreEmptiveTimer1.iState = 0;

cPreEmptiveTimer1.setWaitConst(LED1\_STATE\_UP); // 1 sec

cPreEmptiveTimer2.iState = 0;

cPreEmptiveTimer2.setWaitConst(LED2\_STATE\_UP); // 700 ms)

cPreEmptiveTimer3.iState = 0;

cPreEmptiveTimer3.setWaitConst(LED3\_STATE\_UP); // 650 ms)

cPreEmptiveTimer4.iState = 0;

cPreEmptiveTimer4.setWaitConst(LED4\_STATE\_UP); // 1200 ms)

}

Voor elke state machine kan dus de code herschreven worden; de getallen worden vervangen door de gedefinieerde woorden.

Regel 41 t/m 54 beschrijft de statemachine voor de eerste led (PIN\_EERSTE\_LED)

if(cPreEmptiveTimer1.timeout(currentMillis)) {

switch(cPreEmptiveTimer1.iState) {

case 0:

digitalWrite(PIN\_EERSTE\_LED, HIGH); // LED aan

cPreEmptiveTimer1.iState = 1; // nieuwe status

cPreEmptiveTimer1.setWaitConst(LED1\_STATE\_UP); // wacht ... sec

break;

case 1:

digitalWrite(PIN\_EERSTE\_LED, LOW); // LED uit

cPreEmptiveTimer1.iState = 0; // nieuwe status

cPreEmptiveTimer1.setWaitConst(LED1\_STATE\_DOWN); // wacht ... sec

break;

}

}

De overige drie statemachines zijn op dezelfde manier veranderd. Dit is in de code terug te vinden.

## Een verbetering in de code (CPreEmptiveTimerSimple\_v3.ino)

Voor deze code moet de laatste library (bibliotheek) van CPreEmptiveTimer geïnstalleerd worden. Zie (Her)installatie van de libraries in TipsEnTricks.docx

De library van CPreEmptiveTimer is herschreven. De initialisatie (eerste definitie) van een CPreEmptiveTimer-object <zal nu altijd beginnen in state 0 (en de later beschreven sub state “STATE\_INIT”. Ook zal de state machine direct met state 0 beginnen.

Door deze update zal de setup() functie vereenvoudigd worden:

* iState hoeft niet meer geïnitieerd te worden; deze staat namelijk standaard op 0.
* De functie setWaitConst in de CPreEmptiveTimer-object hoeft ook niet meer geïnitieerd. De statemachine zal direct state 0 uitvoeren.

De setup zal nu zijn:

void setup() {

pinMode(PIN\_EERSTE\_LED, OUTPUT);

pinMode(PIN\_TWEEDE\_LED, OUTPUT);

pinMode(PIN\_DERDE\_LED, OUTPUT);

pinMode(PIN\_VIERDE\_LED, OUTPUT);

}

Minder code zodat er ook minder fouten gemaakt kunnen worden.

## Nog een verbetering (CPreEmptiveTimerSimple\_v4.ino)

Er zijn in de file CPreEmptiveTimerSimple\_v2.ino vier statemachines (4 leds) die onafhankelijk kunnen knipperen. Wanneer alles correct geïmplementeerd is blijkt dat elke led code heeft die vergelijkbaar is met de code van de andere leds.

De code van de Arduino IDE is een vereenvoudigde implementatie van c++ en zoals elke programmeertaal kunnen in c++ ook functies gedefinieerd worden. Functies zijn stukjes code die bepaalde, vaak voorkomende taken, beschrijven. Zodoende hoeven die vaak voorkomende taken maar één keer beschreven te worden.

De precieze uitleg van een functie gaat in deze beschrijving te ver. Maar in de gebruikte code zijn al functies gebruikt. Zo zijn setup() en loop() functies die zelf gemaakt moeten worden binnen de Arduino. Anderzijds hebben we ook al verschillende functies aangeroepen. De functie digitalWrite() is zo een functie en zorgt ervoor dat een bepaalde digitale pin (bijv. PIN\_EERSTE\_LED een bepaalde waarde krijgt zoals HIGH of LOW). De uiteindelijke beschrijving van digitalWrite staat op de pagina

<https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/>

Dus we kunnen een functie maken die één statemachine definieert. Deze wordt genoemd:

handlePreemptive

Met deze functie moeten wel die argumenten meegegeven worden welke noodzakelijk zijn voor het aansturen van één knipper state machine.

De functie begint met de code:

void handlePreemptive(CPreEmptiveTimer \*cPreEmptiveTimer,

unsigned long currentMillis,

int iPinLed,

int iStateUp,

int iStateDown)

De functie heeft de volgende argumenten:

|  |  |  |
| --- | --- | --- |
| Naam | type | beschrijving |
| cPreEmptiveTimer | CPreEmptiveTimer \* | Een pointer naar een CPreEmptiveTimer object. Eigenlijk: een verwijzing naar het geheugen waar een CPreEmptiveTimer object staat. Dit wordt later uitgelegd. |
| currentMillis | unsigned long | De huidige tijd in de loop() verkregen tijd met de functie millis() |
| iPinLed | int | De pinnummer welke aan of uitgezet moet worden. |
| iStateUp | int | De tijd dat de led aan moet zijn (in ms) |
| iStateDown | int | De tijd dat de led uit moet zijn (in ms) |

De type CPreEmptiveTimer \*cPreEmptiveTimer betekent: een verwijzing naar een object. Zoals eenieder weet heeft een computer een geheugen. Dit geheugen is opgedeeld in adressen en het sterretje (\*) betekend niets meer dan het adres en in dit geval een object van het type CPreEmptiveTimer.

Dit object heeft een aantal eigenschappen (properties) en functies (methods). Wanneer een object direct aangeroepen wordt, dan wordt een punt (.) gebruikt zoals in de oorspronkelijke code:

cPreEmptiveTimer1.iState

Echter, wanneer de variabele een adres bevat dan wordt niet de punt maar een pointerverwijzing gebruikt. Dit wordt aangegeven met ‘->’. In dit geval wordt de eigenschap iState benaderd door:

cPreEmptiveTimer->iState

De aanroep waarbij een adres gebruikt wordt is ook anders. In plaats van alleen de naam van de variabele te gebruiken, wordt nu de ‘en’-teken (‘&’) gebruikt zoals:

handlePreemptive(currentMillis, &cPreEmptiveTimer1, …

Dus:

|  |  |
| --- | --- |
| Gebruik variabele bij het aanroepen | &cPreEmptiveTimer1 |
| Declareren van een object referentie in de functie | \*cPreEmptiveTimer |
| Gebruik van een eigenschap of functie met een referentie | cPreEmptiveTimer->… |

Misschien erg lastig op dit moment maar wanneer dit (veel) gebruikt gaat worden dan blijkt het heel eenvoudig te zijn.

Binnen de functie handlePreemptive wordt nu de code geschreven waarbij één led tegelijkertijd behandeld zal worden.

De hele functie is beschreven van regel 29 t/m 48:

void handlePreemptive(CPreEmptiveTimer \*cPreEmptiveTimer,

unsigned long currentMillis,

int iPinLed,

int iStateUp,

int iStateDown) {

if(cPreEmptiveTimer->timeout(currentMillis)) {

switch(cPreEmptiveTimer->iState) {

case 0:

digitalWrite(iPinLed, HIGH); // LED aan

cPreEmptiveTimer->iState = 1; // nieuwe status

cPreEmptiveTimer->setWaitConst(iStateDown); // wacht ... sec

break;

case 1:

digitalWrite(iPinLed, LOW); // LED uit

cPreEmptiveTimer->iState = 0; // nieuwe status

cPreEmptiveTimer->setWaitConst(iStateUp); // wacht ... sec

break;

}

}

}

De eerste statement (regel 38) bevat de if statement:

if(cPreEmptiveTimer->timeout(currentMillis))

Herinner dat dit in de vorige file de statement was:

if(cPreEmptiveTimer1.timeout(currentMillis))

Dus bijna hetzelfde echter in het eerste geval wordt er een verwijzing naar een object gedaan terwijl in het tweede geval wordt het object direct gebruikt.

De overige regels zijn vrijwel gelijk aan de oorspronkelijke code voor elke state machine alleen hier wordt verwezen naar een object i.p.v. het object direct te gebruiken. (‘->’ i.p.v. ‘.’)

De loop functie wordt nu veel eenvoudiger: roep voor elke statemachine de handlePreemptive functie aan. Dit zijn dus 4 aanroepen; voor elke statemachine één.

Dit is geïmplementeerd in regel 50 t/m 57

void loop() {

unsigned long currentMillis = millis();

handlePreemptive(&cPreEmptiveTimer1, currentMillis, PIN\_EERSTE\_LED,

LED1\_STATE\_UP, LED1\_STATE\_DOWN);

handlePreemptive(&cPreEmptiveTimer1, currentMillis, PIN\_TWEEDE\_LED,

LED2\_STATE\_UP, LED2\_STATE\_DOWN);

handlePreemptive(&cPreEmptiveTimer1, currentMillis, PIN\_DERDE\_LED,

LED3\_STATE\_UP, LED3\_STATE\_DOWN);

handlePreemptive(&cPreEmptiveTimer1, currentMillis, PIN\_VIERDE\_LED,

LED4\_STATE\_UP, LED4\_STATE\_DOWN);

}

De code verbeteringen zijn nu behandeld. Aangeraden wordt om wat te spelen met de LEDx\_STATE\_... #define’s. Verschillende waardes zullen de snelheid maar ook de snelheden tussen aan en uit veranderen.

# De verbeterde statemachine

## Aanpassingen voor states en events (CPreEmptiveTimerSimple\_v5.ino)

In dit hoofdstuk worden de states en de events omgedraaid in de code. Het is beter om eerst te bepalen in welke state de statemachine zich bevind om daarna te gaan bepalen of een event heeft plaatsgevonden.

In de onderstaande code is de functie handlePreemptive (zie Nog een verbetering (CPreEmptiveTimerSimple\_v4.ino) aangepast. De inhoud van deze functie (regel 29 t/m 50 is veranderd in:

switch(cPreEmptiveTimer->iState) {

case 0:

if(cPreEmptiveTimer->timeout(currentMillis)) {

digitalWrite(iPinLed, HIGH); // LED aan

cPreEmptiveTimer->iState = 1; // nieuwe status

cPreEmptiveTimer->setWaitConst(iStateUp); // wacht ... sec

}

break;

case 1:

if(cPreEmptiveTimer->timeout(currentMillis)) {

digitalWrite(iPinLed, LOW); // LED uit

cPreEmptiveTimer->iState = 0; // nieuwe status

cPreEmptiveTimer->setWaitConst(iStateDown); // wacht ... sec

}

break;

}

Op regel 34 is nu de switch statement geplaatst waardoor eerst bepaald word in welke state de statemachine zich bevindt. Hierna, na de case statement, wordt naar de event gekeken. In deze code is er maar één event: de timeout, maar dit kunnen er natuurlijk ook meer zijn zoals in de AKI beschreven zal worden.

Na het bepalen of de event is opgetreden, wordt de initialisatie voor de volgende state gedaan. Dit zal in het volgende sub-hoofdstuk verbeterd worden.

## Introductie van de state initialisatie (CPreEmptiveTimerSimple\_v6.ino)

Onderstaande code welke de initialisatie acties en event acties beschrijft ziet ongeveer als volgt uit:

switch(cPreEmptiveTimer->iState) {

case 0:

switch(cPreEmptiveTimer->iSubState) {

case STATE\_INIT:

// initialisatie

break;

case STATE\_WAIT:

break;

}

}

Noot: in de pseudo code van 02\_PreemptiveStateCode.docx wordt een ALS (if) statement in de pseudo gebruikt om de iSubState te bepalen en in deze code staat hier een switch. Het is normaal dat voor statemachines de switch gebruikt wordt.

Nu moet alleen de initialisatie en de event afhandeling binnen de code gezet worden. Dit is vooral kopieer en plak van de oorspronkelijke code.

De initialisatie voor een led aan is:

digitalWrite(iPinLed, HIGH); // LED aan

cPreEmptiveTimer->setWaitConst(iStateUp); // wacht ... sec

cPreEmptiveTimer->iSubState = STATE\_WAIT; // nieuwe sub status

break;

1. De led wordt eerst aangezet
2. Dan wordt de tijd bepaald hoelang de led aan moet blijven
3. En als laatste zal de volgende sub state gezet worden.

De event afhandeling is:

if(cPreEmptiveTimer->timeout(currentMillis)) { // Is event opgetreden?

// Zet de nieuwe state

// setNewState() zet automatisch iSubState = STATE\_WAIT

cPreEmptiveTimer->setNewState(1);

}

break;

Dit is eigenlijk heel eenvoudig:

1. Wacht totdat de event (in dit geval een timeout) is opgetreden.
2. Indien zo zet de nieuwe state.

Noot: bij punt 2:

De setNewState functie doet 2 dingen:

* Het zet iState op de opgegeven waarde
* De functie zal iSubState op de waarde STATE\_INIT zetten

Deze regel kan vervangen worden door:

cPreEmptiveTimer->iState = 1;

cPreEmptiveTimer->iSubState = STATE\_INIT;

Het is dus eenvoudiger om de setNewState functie te gebruiken.

# De AKI

“De modernere AKI, vanaf 1955, bevat drie lampen, twee rode (onder) en een witte (boven). De twee rode gaan snel knipperen als er een trein in aantocht is. Als de rode lichten uit zijn knippert het witte licht rustig, ten teken dat de overweg veilig is om over te steken.”

In dit hoofdstuk wordt de AKI als basis gebruikt om de verschillende stappen van de implementatie van een state machine uit te leggen. Om de states grafisch weer te geven gebruik ik de web site:

<https://www.cs.unc.edu/~otternes/comp455/fsm_designer/>

Voor de AKI is de volgende state diagram gemaakt:

Afbeelding met lijn, tekst, diagram, cirkel

Automatisch gegenereerde beschrijving

Met de bijbehorende tabel:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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 |  |

Afbeelding met diagram, stroomkring

Automatisch gegenereerde beschrijvingDe bijbehorende Frizing schakeling is:

Hierin zijn 2 rode leds, één witte led voor de AKI te maken.

De baan bezet zullen we simuleren door een drukknop.

De Friyzing schema is opgeslagen in de file AKI.fzz

### Beschrijving van de verschillende states

Voor elke statemachine kunnen één of meerdere states gedefinieerd worden. Voor de beschrijving van een AKI zijn vier states te onderscheiden:

1. Rode lichten zijn uit en wit licht is aan (state 0)
2. Rode lichten zijn uit en wit licht is uit (state 1)
3. Linker rood licht aan, rechter rood licht uit en wit licht uit (state 2)
4. Linker rood licht uit, rechter rood licht aan en wit licht uit (state 3)

### Beschrijving van de initialisatie en wachten op een event van de state

Voor de code worden gebruik gemaakt van de volgende #defines:

Voor de leds:

#define PIN\_ROOD\_LINKS 11

#define PIN\_ROOD\_RECHTS 10

#define PIN\_WIT 6

De drukknop:

#define PIN\_BUTTON 5

En de frequentie van de knipperlichten:

#define WIT\_KNIPPER\_FREQ 750 // 2x 3/4 sec

#define ROOD\_KNIPPER\_FREQ 500 // 2x een halve sec

In de code hebben we maar één CPreEmptiveTimer object nodig. Om ervoor te zorgen dat de timing bij een state wisseling goed verloopt wordt geen gebruik gemaakt van de setWaitConst functie maar van de setWait functie. Dit vereist een iets meer tikwerk maar zorgt voor dat de time-out tijd preciezer wordt bepaald.

Noot: het verschil tussen de setWaitConst() en de setWait() functie is:

De setWait functie krijgt als parameter

#### State 0

Initialisatie bestaat uit het aanzetten van een de witte led.

Hierna wordt gewacht op één van de twee events.

|  |  |
| --- | --- |
| Baanvak\_bezet: | Deze heeft de hoogste prioriteit. Wanneer dit gebeurd, zal de witte led worden uitgezet. Vervolgens zal de volgende state 2 worden. |
| Timeout\_wit: | de tijd dat de witte led aan moet staan is verstreken. De volgende state zal 1 worden. |

De bijbehorende code is:

case 0: // Witte led aan

switch(cPreEmptiveTimer.iSubState) {

case STATE\_INIT:

digitalWrite(PIN\_WIT, HIGH); // LED aan

cPreEmptiveTimer.setWait(currentMillis, WIT\_KNIPPER\_FREQ); // wacht ... sec

cPreEmptiveTimer.iSubState = STATE\_WAIT; // nieuwe sub status

break;

case STATE\_WAIT:

if(val==HIGH) {

// Een trein is in het blok

digitalWrite(PIN\_WIT, LOW);

cPreEmptiveTimer.setNewState(2); // de rode leds moeten gaan knipperen

} else if(cPreEmptiveTimer.timeout(currentMillis)) {// Timeout opgetreden?

// Zet de nieuwe state

// setNewState() zet automatisch iSubState = STATE\_INIT

cPreEmptiveTimer.setNewState(1);

}

break;

}

break;

#### State 1

Initialisatie bestaat uit het uitzetten van een de witte led.

Hierna wordt gewacht op één van de twee events.

|  |  |
| --- | --- |
| Baanvak\_bezet: | Deze heeft de hoogste prioriteit. Wanneer dit gebeurd, zal de volgende state 2 worden. De witte led hoeft niet uitgezet te worden omdat deze al uit is. |
| Timeout\_wit: | de tijd dat de witte led aan moet staan is verstreken. De volgende state zal 0 worden. |

De bijbehorende code is:

case 1: // Witte led aan

switch(cPreEmptiveTimer.iSubState) {

case STATE\_INIT:

digitalWrite(PIN\_WIT, LOW); // LED uit

cPreEmptiveTimer.setWait(currentMillis, WIT\_KNIPPER\_FREQ); // wacht ... sec

cPreEmptiveTimer.iSubState = STATE\_WAIT; // nieuwe sub status

break;

case STATE\_WAIT:

if(val==HIGH) {

// Een trein is in het blok

// digitalWrite(PIN\_WIT, LOW); is niet nodig; witte led is al uit

cPreEmptiveTimer.setNewState(2); // de rode leds moeten gaan knipperen

} else if(cPreEmptiveTimer.timeout(currentMillis)) { // Timeout opgetreden?

// Zet de nieuwe state

// setNewState() zet automatisch iSubState = STATE\_INIT

cPreEmptiveTimer.setNewState(0);

}

break;

}

break;

#### State 2

Initialisatie bestaat uit

1. Het aanzetten van een de linker rode led.
2. Het uitzetten van de rechter rode led

Hierna wordt gewacht op één van de twee events.

|  |  |
| --- | --- |
| Baanvak\_onbezet. | Deze heeft de hoogste prioriteit. Wanneer dit gebeurd, zal de linker led uitgezet worden. De rechter led hoeft niet uitgezet te worden omdat deze al uit is. |
| Timeout\_rood: | de tijd dat een rode led aan moet staan is verstreken. De volgende state zal 3 worden. |

De bijbehorende code is:

case 1: // Witte led aan

switch(cPreEmptiveTimer.iSubState) {

case STATE\_INIT:

digitalWrite(PIN\_WIT, LOW); // LED uit

cPreEmptiveTimer.setWait(currentMillis, WIT\_KNIPPER\_FREQ); // wacht ... sec

cPreEmptiveTimer.iSubState = STATE\_WAIT; // nieuwe sub status

break;

case STATE\_WAIT:

if(val==HIGH) {

// Een trein is in het blok

// digitalWrite(PIN\_WIT, LOW); is niet nodig; witte led is al uit

cPreEmptiveTimer.setNewState(2); // de rode leds moeten gaan knipperen

} else if(cPreEmptiveTimer.timeout(currentMillis)) { // Timeout opgetreden?

// Zet de nieuwe state

cPreEmptiveTimer.setNewState(0);

}

break;

}

break;

#### State 2

Initialisatie bestaat uit

1. Het uitzetten van een de linker rode led.
2. Het aanzetten van een de rechter rode led.

Hierna wordt gewacht op één van de twee events.

|  |  |
| --- | --- |
| Baanvak\_onbezet. | Deze heeft de hoogste prioriteit. Wanneer dit gebeurd, zal de rechter led uitgezet worden. De linker led hoeft niet uitgezet te worden omdat deze al uit is. |
| Timeout\_rood: | de tijd dat een rode led aan moet staan is verstreken. De volgende state zal 2 worden. |

De bijbehorende code is:

case 3: // Links rood aan, rechts rood uit

switch(cPreEmptiveTimer.iSubState) {

case STATE\_INIT:

digitalWrite(PIN\_ROOD\_LINKS, LOW); // linker LED uit

digitalWrite(PIN\_ROOD\_RECHTS, HIGH); // rechter LED aan

cPreEmptiveTimer.setWait(currentMillis, ROOD\_KNIPPER\_FREQ); // wacht ... sec

cPreEmptiveTimer.iSubState = STATE\_WAIT; // nieuwe sub status

break;

case STATE\_WAIT:

if(val==LOW) {

// De trein is uit het blok

digitalWrite(PIN\_ROOD\_RECHTS, LOW); // links is al uit

cPreEmptiveTimer.setNewState(0); // state 0: witte led knipperen

} else if(cPreEmptiveTimer.timeout(currentMillis)) { // Timeout opgetreden?

cPreEmptiveTimer.setNewState(2);

}

break;

}

break;

# Van de originele

**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 waarbij 4 ledjes onafhankelijk van elkaar gaan knipperen.. 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