EMBEDDED SOFTWARE DEVELOPMENT (ESD)

LES 1-3: FRITZING EN LEDPATROON IN ARRAY (VOORBEREIDING)

Versie: 13-2-2017 13:19:00

INHOUDSOPGAVE

1 FRITZING

Fritzing is een programma gemaakt door Fachhochschule Potsdam. Je kunt het downloaden via http://fritzing.org/download/

Met Fritzing kun je een beschrijving maken van de aansluitingen van een Arduino-schakeling. Veel voorbeelden met de Arduino zijn te vinden op internet in de vorm van een Fritzing-tekening. Vanaf nu ga ook jij Fritzing gebruiken om ontwerpen te maken van je hardwareopstelling, voor je deze daadwerkelijk in elkaar zet. Op deze manier wordt je gedwongen eerst na te denken. Bovendien kun je bij twijfel hulp inroepen van een collega-student of je docent, voordat je je fysieke opstelling hebt gebouwd.

2 SOFTWAREARCHITECTUUR

In de eerste les hebben we kennis gemaakt met de software architectuur, het verdelen van toen nog de knipperende led over twee tabbladen.

Wellicht ben je in de tweede les aan de slag gegaan met dezelfde architectuur en heb je een stuk code ontwikkeld zoals hieronder (dit is de halve oplossing van een led heen en weer laten branden):

main.ino	LEDcontrol.ino

`const int NROFLEDS = 8;`
`const int DELAYTIME = 500;`
 	
`for (i=0; i < NROFLEDS; i++) {`
`  ledControlLEDOn(i);`
`  delay(DELAYTIME);`
`  ledControlLEDOff(i);`
`}`
	`int pinArray[NROFLEDS] = {13, 12, 11, 10, 9, 8, 7, 6};`
	
	`void ledControlLEDOn(int led) {`
	`……`
	`}`

We hebben twee verbetervoorstellen voor bovenstaande code:

- Het hoofdprogramma bevat ook geen patronen meer
- De code voor het maken van de patronen moet onafhankelijk van het aantal leds gemaakt worden

Allereerst maken we een extra tabblad met daarin functies, die de led-patronen maken:

main.ino	patronen.ino	LEDControl.ino

`void loop() {`
 	
`……`
`patroonHeen();`
 	
`}`
	`const int NROFLEDS = 8;`
	`const int DELAYTIME = 500;`
	
	`void patroonHeen {`
	`  for (i=0; i < NROFLEDS; i++) {`
	`  ledControlLEDOn(i);`
	`  delay(DELAYTIME);`
	`  ledControlLEDOff(i);`
	`  }`
	`}`
		`int pinArray[NROFLEDS] = {13, 12, 11, 10, 9, 8, 7, 6};`
		
		`void ledControlLEDOn(int led) {`
		`……`
		`}`

Het hoofdprogramma bevat nu veel minder code, maar die gingen we nog uitbreiden met code om een commando in te lezen welk patroon we gaan laten zien. De patronen zitten nu netjes bij elkaar in een apart bestand.

Maar we willen ook, dat het tabblad met patronen niet 'weet' hoeveel leds er zijn; de hoeveelheid leds is een hardware-afhankelijk iets en alle hardware-afhankelijk zaken stoppen we in LEDControl. Probleem is echter dat patronen.ino toch het aantal leds moet 'weten' om een mooi patroon te maken. Deze informatie 'haalt' hij uit LEDControl middels een aparte functie getNrLeds():

main.ino	patronen.ino	LEDControl.ino

`void loop() {`
 	
`……`
`patroonHeen();`
 	
`}`
	`const int DELAYTIME = 500;`
	
	`void patroonHeen {`
	`  for (i=0; i < getNrLeds(); i++) {`
	`  ledControlLEDOn(i);`
	`  delay(DELAYTIME);`
	`  ledControlLEDOff(i);`
	`  }`
	`}`
		`const int NROFLEDS = 8;`
		`int pinArray[NROFLEDS] = {13, 12, 11, 10, 9, 8, 7, 6};`
		
		`int getNrLeds() {`
		`  return NROFLEDS;`
		`}`
		
		`void ledControlLEDOn(int led) {`
		`……`
		`}`

Later zullen bij de course 'Object Oriented Program Development' dit soort constructies van zogenaamde "getter"-functies worden behandeld.

3 MEERDERE LEDS IN ARRAY

Gegeven onderstaande sketch (die niet voldoet aan onze coderingsafspraken):

```
int pinArray[8] = {13, 12, 11, 10, 9, 8, 7, 6};
int LEDsequence [8][8] = {
  {1,0,0,0,0,0,0,0},
  {0,1,0,0,0,0,0,0},
  {0,0,1,0,0,0,0,0},
  {0,0,0,1,0,0,0,0},
  {0,0,0,0,1,0,0,0},
  {0,0,0,0,0,1,0,0},
  {0,0,0,0,0,0,1,0},
  {0,0,0,0,0,0,0,1}
};
int sequence;
int pin;

void setup(){
  // we make all the declarations at once
  for (pin=0;pin<8;pin++) {
    pinMode(pinArray[pin], OUTPUT);
```

```
  }
}

void loop() {
  for (sequence=0;sequence<8;sequence++) {
    for (pin=0;pin<8;pin++) {
     digitalWrite(pinArray[pin], LEDsequence[sequence][pin]);
    }
   delay(100);
 }
}
```

3.1 LEDPATROON IN ARRAY

Een belangrijke vraag is nu: welke van de twee oplossingen (dus een patroon in code of een patroon in arrays) heeft de voorkeur.

Het antwoord is flauw: dat ligt eraan. De snelheid, waarmee een programmeur de ledpatronen kan wijzigen in een oplossing met arrays is veel groter dan bij een oplossing in code.

Maar… de oplossing met arrays kost meer geheugen en zoals we verderop in het materiaal zullen leren, heeft de Arduino niet superveel geheugen. Als we te veel patronen willen opslaan lukt dat niet.