# Het Metronom real-time besturingssysteem

## een RTOS voor AVR-processoren

**Dieter Profos, dr. sc. techn. ETH (Zwitserland)**

Voor veel taken – zoals het verwerken van continue signalen – moeten microcontrollers taken uitvoeren met een exacte timing. Het hier gepresenteerde real-time besturingssysteem is (ook) geschikt voor AVR-controllers met weinig geheugen. Je moet enkele beperkingen accepteren, zoals puur in assembler programmeren, maar dat is nog altijd een goed compromis voor projecten waar snelheid en real-time mogelijkheden belangrijk zijn.

### Waarom wéér een besturingssysteem?

Met het verschijnen van kleine en zeer kleine processoren of controllers werden processen automatiseerbaar waarvoor het gebruik van een 'echte' computer vroeger niet te rechtvaardigen was. Deze microcontrollers hoeven geen periferie aan te sturen (toetsenbord, muis, scherm, harde schijf enzovoort), zodat de besturingssystemen kunnen worden teruggebracht tot wat strikt noodzakelijk is om de verwerking van gebruikersprogramma's te organiseren.

De meeste besturingssystemen zijn ontworpen om zoveel mogelijk programma's zo efficiënt mogelijk en (vanuit het oogpunt van de gebruiker) gelijktijdig te laten draaien. De situatie is echter anders wanneer continue, tijdgebonden signalen moeten worden verwerkt: dit vereist processen die met een exacte timing verlopen. De `delay()` functie in Arduino is dan bijvoorbeeld niet meer nauwkeurig genoeg, omdat deze alleen wachttijden genereert zonder rekening te houden met de voor de verwerking benodigde runtimes, wat duidelijk merkbaar wordt bij samplingtijden van 1 ms of korter.

Daarom moeten de volgende twee problemen worden opgelost:

- Bepaalde taken moeten precies op vooraf bepaalde tijden worden uitgevoerd, andere alleen als er tijd voor over is.
- Elke onderbreekbare taak heeft zijn eigen stack nodig voor het bufferen van de registerinhoud. Bij kleine controllers is de geheugenruimte echter vrij beperkt: bijvoorbeeld 750 bytes bij de ATtiny25 of 1 K bij de ATmega8.

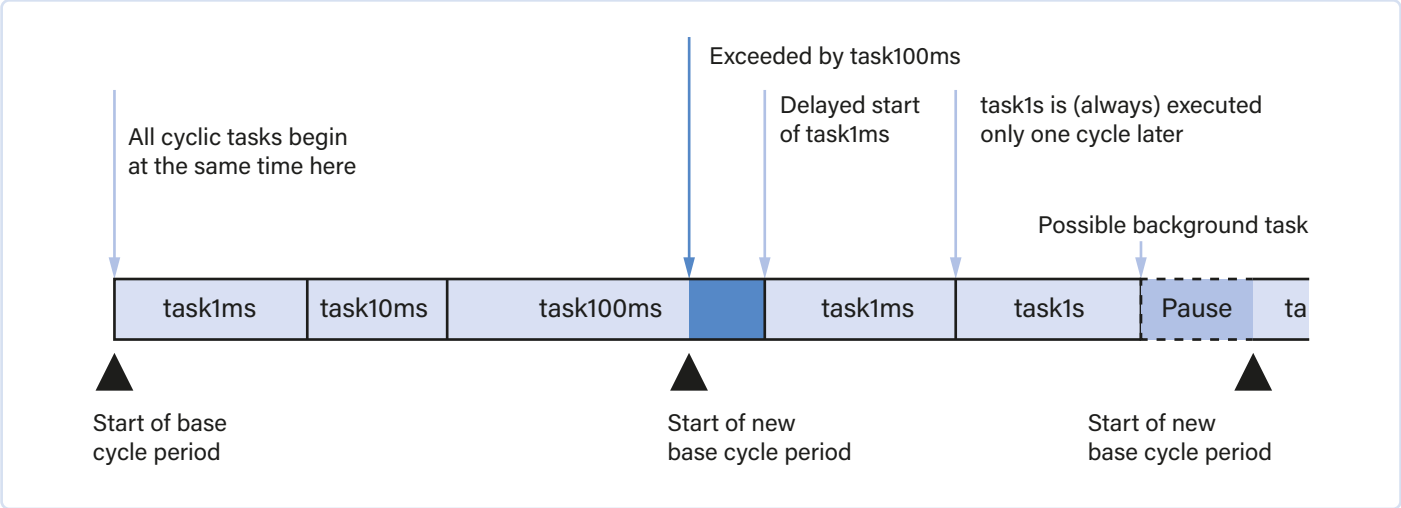
Het hier gepresenteerde besturingssysteem *Metronom* kan worden gedownload van de Elektor-website [1] als open-source software onder de BSD-2 licentie; een release via GitHub staat voor de komende maanden op de planning.

### Cyclische taken

Metronom is speciaal ontworpen voor de uitvoering van zogenaamde *cyclische* taken met nauwkeurig bepaalde tijdsintervallen (met maximaal 8 verschillende cyclustijden). Er is precies één taak per cyclustijd; als meerdere activiteiten met onafhankelijke inhoud in dezelfde cyclus moeten worden uitgevoerd, moeten ze in dezelfde taak worden gecombineerd.

De cyclustijden worden als volgt gegenereerd:

- de basis-cyclusperiode (bijvoorbeeld 1 ms) wordt vastgelegd met behulp van de processorhardware (kristal of interne RC-oscillator, hardware- en interruptgestuurde softwareteller), en
- de andere cyclustijden worden gegenereerd door een keten van




*Figuur 1. Een opeenvolging van verschillende cyclische taken (extreem geval).*

tellers, en wel zo dat elke cyclusperiode een veelvoud is van de vorige (de standaardinstelling is bijvoorbeeld 1 ms → 10 ms → 100 ms → 1 s).

Een belangrijk kenmerk van het hier gepresenteerde besturingssysteem is dat cyclische taken elkaar niet kunnen onderbreken (*non-preemptive*). Enerzijds zorgt dit ervoor dat de timing van deze taken zo nauwkeurig mogelijk is, en anderzijds verliest de processor geen 'onproductieve tijd' door taakwisseling. En aangezien elke cyclische taak – als die eenmaal gestart is – zonder onderbreking (behalve door interrupts) wordt voltooid voordat de volgende cyclische taak wordt gestart, kunnen alle cyclische taken samen dezelfde stack gebruiken.
Maar wat gebeurt er als een taak langer loopt dan de basis-cyclustijd (wat eigenlijk door de programmeur vermeden moet worden) of als er meerdere cyclische taken in dezelfde basiscyclus zijn gestart, waarbij de som van de runtimes de basis-cyclustijd overschrijdt (wat zeker legitiem is)? Hier komt een andere eigenschap van Metronom tot uiting: de cyclische taken hebben prioriteiten. De taak met de kortste cyclustijd heeft de hoogste prioriteit, de 'op één na snelste' taak heeft de op één na hoogste prioriteit – enzovoort. Als niet alle gelijktijdig gestarte cyclische taken binnen de basis-cyclustijd kunnen worden afgewerkt, wordt de lopende taak na het verstrijken van de basis-cyclustijd voortgezet – maar dan wordt eerst de 'snelste' taak met de hoogste prioriteit opnieuw uitgevoerd, en pas daarna worden andere eerder gestarte cyclische taken opnieuw uitgevoerd.
Een voorbeeld: de implementatie van de cyclustijden zorgt ervoor dat alle cyclische taken elke seconde gelijktijdig worden gestart. Wat er dan gebeurt is te zien in **figuur 1**.
Dit betekent dat, als absolute bovengrens, elke cyclische taak niet langer mag duren dan de kortste cyclustijd – dat wil zeggen 1 ms in ons voorbeeld. Dit komt overeen met ongeveer 6000 instructies bij een ATtiny op 8 MHz en ongeveer 14000 instructies bij een ATmega op 16 MHz (de rest van de instructies wordt – gemiddeld – gebruikt door het besturingssysteem zelf en voor de afhandeling van interrupts).

## Achtergrondtaken

Er zijn echter bepaalde bewerkinge die door hun aard langer duren:

- schrijven naar EEPROM duurt bijvoorbeeld enkele milliseconden (meestal ongeveer 3,3 ms), dat wil zeggen een onacceptabel lange tijd voor een basiscyclus van 1 ms;
- tekst versturen met 9600 bd is niet haalbaar met een basiscyclus van 1 ms, omdat zelfs de verzending van één enkel teken al langer dan 1 ms duurt;
- wanneer langere berekeningen (bijvoorbeeld geëmuleerde rekenkundige bewerkingen!) moeten worden uitgevoerd of tekenreeksen moeten worden verwerkt, duurt dit vaak te lang binnen een cyclische taak en worden de tijdgebonden processen geblokkeerd.

Dit betekent dat er nog een manier moet zijn om dergelijke processen te delegeren naar een soort onderbreekbare taak. Hiervoor wordt een combinatie van twee methoden gebruikt:

- het gebruik van interrupts in plaats van actief wachten: hierdoor kan het wachten op het einde van een bewerking (bijvoorbeeld het verzenden van een teken) worden 'gedelegeerd' aan de hardware; deze methode wordt gebruikt voor interruptgestuurde bewerkingen. Dit lost de problemen op voor de overdracht van een enkel teken of voor het schrijven van een enkele waarde naar de EEPROM, maar niet voor het wachten op het einde van de complete bewerking (zoals het verzenden van een volledige tekst);
- uitvoering van achtergrondtaken: een achtergrondtaak loopt alleen gedurende de tijd die niet door cyclische taken wordt bezet. Bovendien kan hij op elk moment worden onderbroken, zodat hij de tijdige uitvoering van de cyclische taken niet verstoort.

Als een achtergrondtaak echter eenmaal loopt, kan deze niet meer door andere achtergrondtaken worden onderbroken. Er wordt dus maar één achtergrondtaak tegelijk uitgevoerd, en als die moet wachten, wacht de hele verwerking van de achtergrondtaken. Hoewel dit de verwerking van de achtergrondtaken vertraagt, betekent het dat slechts één stackgedeelte hoeft te worden gereserveerd voor alle achtergrondtaken.
Achtergrondtaken worden gekenmerkt door de volgende eigenschappen:

- een achtergrondtaak kan te allen tijde worden onderbroken ten gunste van cyclische taken, maar niet ten gunste van een andere achtergrondtaak;
- de uitvoering van een achtergrondtaak wordt gestart door een startoproep aan de dispatcher;

- achtergrondtaken worden na elkaar uitgevoerd in de volgorde waarin ze zijn gestart;
- achtergrondtaken kunnen wachten op gebeurtenissen (`WAIT_EVENT`), die bijvoorbeeld door interrupt-gestuurde processen worden getriggerd;
- achtergrondtaken kunnen ook wachten op vooraf gedefinieerde tijden (`DELAY`);
- elke achtergrondtaak kan een startbericht van drie 16bit-woorden krijgen, waarmee zijn taak kan worden gespecificeerd (een vierde woord is gereserveerd voor het startadres van de taak);
- binnen het gebruikersprogramma kan een willekeurig aantal achtergrondtaakroutines bestaan; er kunnen er echter maximaal acht tegelijk worden gestart.

De coördinatie van de taken onderling ('Wanneer mag welke taak draaien?') wordt afgehandeld door wat de *dispatcher* wordt genoemd. Deze voert alle 'administratieve processen' uit, zoals het starten van taken, het in veiligheid brengen/herstellen van de processorregisters of het in- en uitschakelen van interrupts.

## Exceptions

Aangezien microcontrollers meestal geen tekstgeoriënteerde periferie hebben, is debuggen erg moeilijk, vooral voor tijdsafhankelijke functies, aangezien breakpoints en dergelijke de timing volledig verstoren. Daarom biedt de kernel van het besturingssysteem een vereenvoudigd mechanisme voor het afhandelen van exceptions, dat in twee fasen is verdeeld:

- een globaal `try-catch` gebied vangt alle uitzonderingen (exceptions/errors) op die zich voordoen in de kernel en in de rekenkundige emulaties. De uitzonderingsspecifieke gegevens kunnen worden opgeslagen in de EEPROM en/of uitgevoerd via USART; daarna voert het besturingssysteem een totale systeem-RESET uit (inclusief `user-reset`). Dit exception-gebied is altijd actief;
- daarnaast kan een toepassingsgericht `try-catch` gebied worden gebruikt, dat alleen betrekking heeft op het eigenlijke gebruikersprogramma. De afhandeling van dergelijke exceprions is in eerste instantie hetzelfde als hierboven: de betreffende gegevens worden opgeslagen en/of uitgevoerd via USART; vervolgens wordt een door de gebruiker op te geven 'application restart'-routine uitgevoerd (subroutine `user_restart`).

## Interruptverwerking

Interrupts worden op vier verschillende manieren behandeld:

- de reset-interrupt wordt gebruikt door het besturingssysteem en is niet direct toegankelijk voor de gebruiker. Omdat de gebruiker deze interrupt echter ook nodig heeft om zijn eigen processen te initialiseren, roept het besturingssysteem na zijn eigen initialisatie de subroutine `user_init` aan, die de gebruiker kan vullen met zijn applicatiespecifieke initialisatiecode;
- timer/counter0 wordt gebruikt voor het genereren van de basisklok voor alle cyclische processen; hij is daarom niet toegankelijk voor de gebruiker;
- voor het gebruik van de EEPROM en de USART biedt het besturingssysteem kant-en-klare driverblokken, die tijdens het genereren van het besturingssysteem kunnen worden geïntegreerd (zie hieronder). De gebruiker kan echter ook zijn eigen serviceroutines aan deze interrupts koppelen of ze gewoon open laten wanneer ze niet worden gebruikt;
- alle andere interrupts zijn direct beschikbaar voor de gebruiker. Voor elke interrupt moet een interrupt service routine en een interrupt initialisatie routine worden gespecificeerd; als meer dan één interrupt bij een apparaat hoort (bijvoorbeeld timers of USART), is een gedeelde initialisatieroutine voldoende. Hiervoor activeert de gebruiker de overeenkomstige parameters in het generatorbestand en voegt de inhoud van de relevante initialisatie- en serviceroutines in zijn gebruikersprogramma in;
- ongebruikte interrupts worden automatisch 'onderschept' door het besturingssysteem.

## Programmeeromgeving

Om efficiencyredenen is Metronom geschreven in AVR assembler (Atmel/Microchip) en gaat er dus van uit dat gebruikersprogramma's ook in assembler zijn geschreven; een interface naar C is niet geïmplementeerd. Wel is er een bibliotheek met vele subroutines, bijvoorbeeld voor 8-bit en 16-bit rekenen (vier rekenkundige basisbewerkingen); een 16-bit breukenbibliotheek is in voorbereiding.
Om het programmeerwerk te vergemakkelijken, zijn alle aanroepen van het besturingssysteem beschikbaar als macro's. Om benamingsbotsingen te voorkomen, geldt de volgende naamgevingsconventie: alle variabelen en jump targets binnen het besturingssysteem en de bibliotheken beginnen met een underscore ("_") – daarom mogen alle namen in het gebruikersprogramma alleen met letters beginnen. Andere tekens dan letters, cijfers en de underscore zijn niet toegestaan.
De algemene structuur van Metronom en het bijbehorende gebruikersprogramma is te zien in **figuur 2**.

## Aanroepen van het besturingssysteem

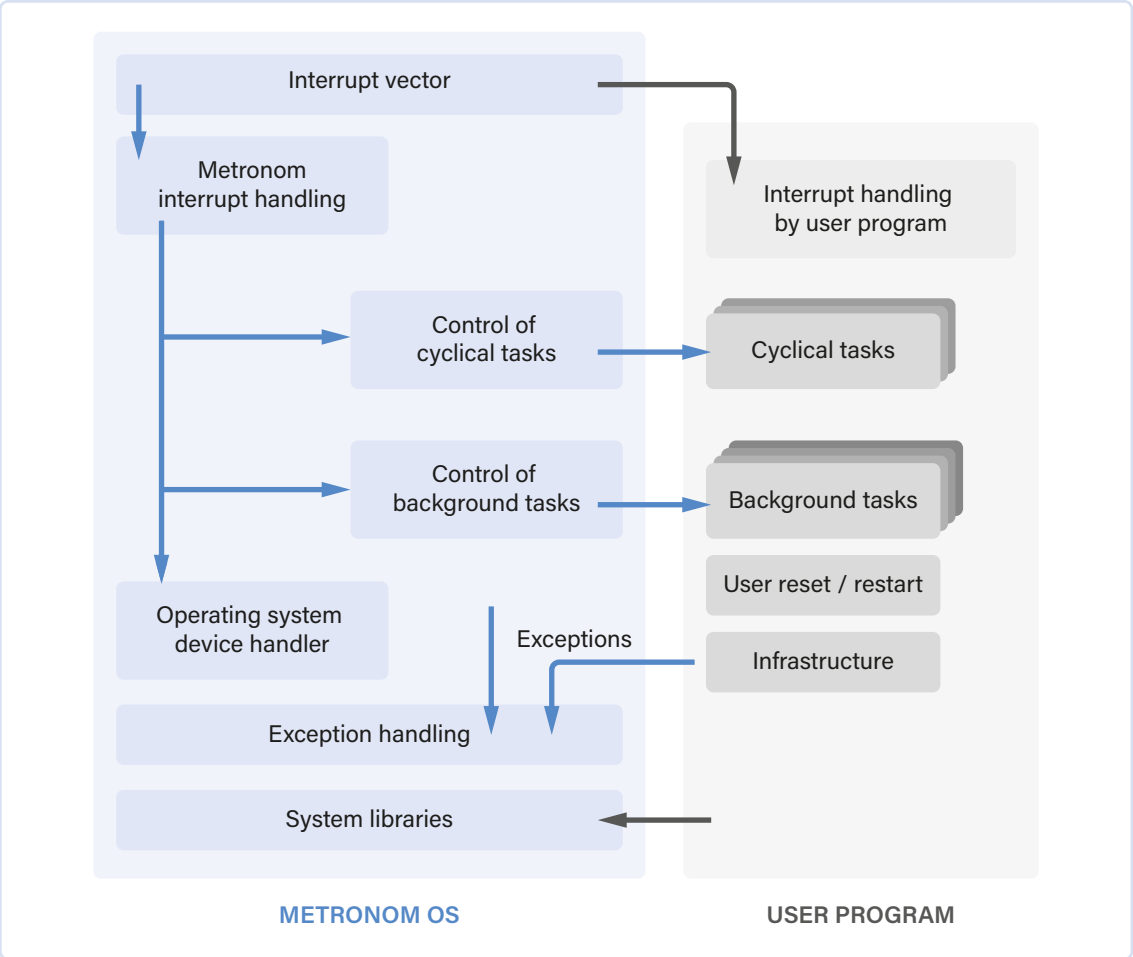
Voor de volledige lijst van besturingssysteem-aanroepen wordt verwezen naar de referenties aan het eind van het artikel; hier volgt slechts een ruw overzicht:

**Macro's voor het afhandelen van exceptions**

- `KKTHROW` zorgt voor een systeembrede exception, dat wil zeggen na het opslaan/uitvoeren van de exception-informatie wordt het hele systeem opnieuw opgestart;
- `KTHROW` zorgt voor een exception die beperkt is tot het gebruikersprogramma, dat wil zeggen na het opslaan/uitvoeren van de exception-informatie wordt alleen de gebruikerssubroutine `user_restart` uitgevoerd; daarna worden de cyclische taken opnieuw gestart.

**Macro's voor het gebruik van achtergrondtaken**

- `_KSTART_BTASK` start een achtergrondtaak;
- `_KDELAY` brengt de aanroepende achtergrondtaak in slaap gedurende n (0 tot 65535) ms;
- `_KWAIT` brengt de aanroepende achtergrondtaak in slaap, waaruit hij kan worden gewekt door middel van...
- `_KCONTINUE`.




*Figuur 2. Algemene structuur van Metronom en het gebruikersprogramma.*

**Macro's voor 8-bit en 16-bit rekenen**

Over het algemeen worden voor rekenkundige bewerkingen van allerlei aard de registers r25:r24 gebruikt als accumulator en r23:r22 als geheugen voor de tweede operand (indien nodig). Voor dit doel zijn meer dan 20 verschillende functies beschikbaar, zoals `_mul8u8` voor een 8×8-bit vermenigvuldiging of `_abs16` voor een 16-bit absolute waarde. Verder zijn er veel pseudocodes voor laden en opslaan, zoals `_ld16` (16-bit getal in de accumulator laden).

**Macro's voor EEPROM-gebruik**

- `_KWRITE_TO_EEPROM` voor het schrijven naar EEPROM;
- `_KREAD_FROM_EEPROM` voor het uitlezen van EEPROM.

**Macro's voor USART-gebruik**

- `_KWRITE_TO_LCD` is een specifiek USART-stuurprogramma, dat de benodigde stuurkarakters voor een 2×16 LC-display toevoegt aan de weer te geven tekst;
- `_KREAD_FROM_USART` (nog niet geïmplementeerd).

## Systeemgenerator SysGen

Voor het genereren van een systeem (dat wil zeggen de volledige code) wordt een speciale systeemgenerator SysGen gebruikt, die ook deel uitmaakt van het package. SysGen is niet beperkt tot Metronom, maar kan ook worden gebruikt voor algemene generatortaken.

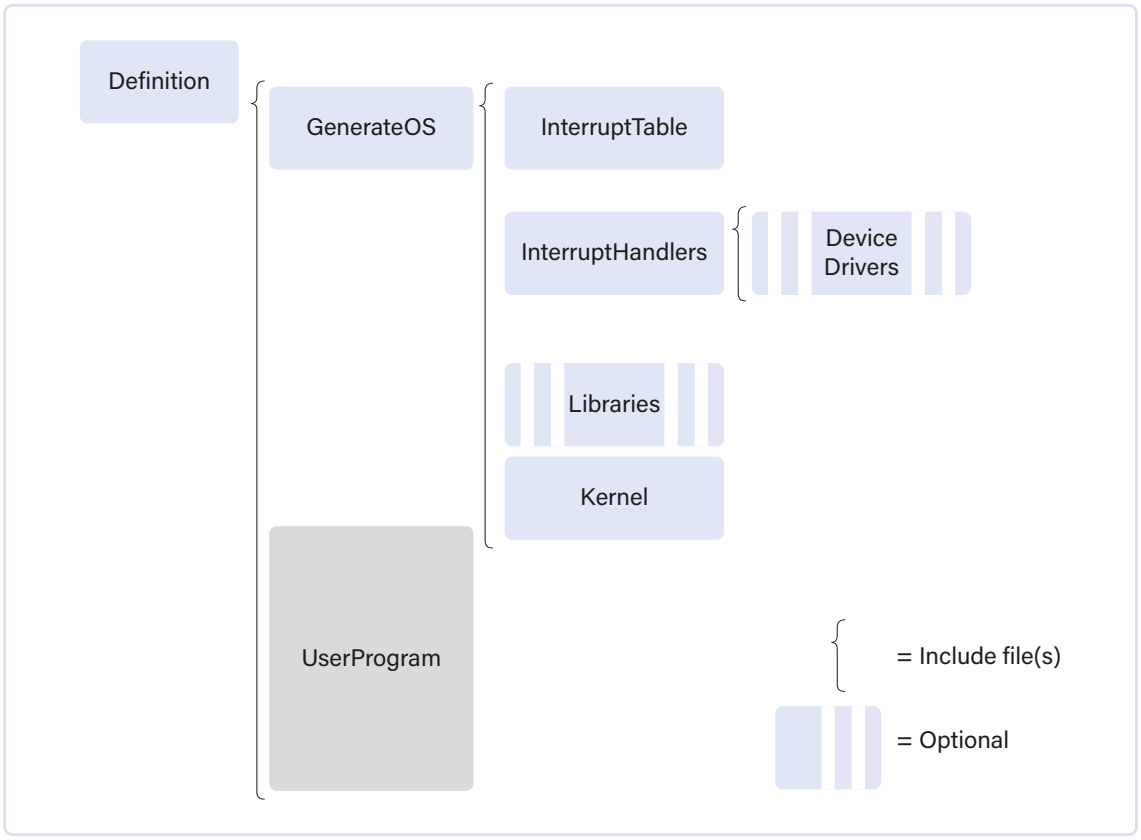
Sommige lezers vragen zich misschien af waarom er een aparte systeemgenerator is ontwikkeld, aangezien er een grote verscheidenheid aan preprocessoren en macrogeneratoren bestaat. Maar voor het genereren van het Metronom-besturingssysteem zijn de functionaliteiten van de preprocessoren in Atmel Studio en standaard C niet voldoende. Met name omdat de preprocessor geen string-'arithmetic' ondersteunt, is het niet mogelijk een 'default directory' of 'library directory' op te geven en van daaruit bestanden te selecteren die zich daarin bevinden. Een zoektocht op *Stack Overflow* toonde aan dat andere mensen hetzelfde probleem hebben als ik, maar geen van de huidige preprocessoren kan ermee overweg.

De preprocessor van Atmel Studio (net als de GNU-preprocessor) biedt de volgende functies voor het samenstellen van de vereiste bestanden:

- `define / set <name> = <numeric_expression>`
- `if … elif … else … endif`, ook genest
- `ifdef, ifndef`
- `include <path> | exit`

De volgende functionaliteiten ontbreken:

- `<path>` kan alleen worden doorgegeven als een vaste string, maar een stringuitdrukking van (een willekeurig aantal) partiële strings, zowel stringvariabelen als stringconstanten, zou nodig zijn;
- `define` en `set` kunnen alleen numerieke waarden toekennen, geen strings, geen aaneenschakeling van strings, en ook geen logische uitdrukkingen;
- voor de (eenmalige) opname van bibliotheekprogramma's zijn de macromogelijkheden in AVRASM of de C-preprocessor niet voldoende; aangezien macro's in AVRASM geen include-statements mogen bevatten, is de automatische opname van bijvoorbeeld emulatieroutines niet mogelijk.




*Figuur 3. Generatorstructuur van Metronom-systemen.*

Dit leidt tot de volgende reikwijdte van de functies:

- `define / set <name> = <numeric_expression> | <string expression> | <boolean expression>`
- `if ... elif ... else ... endif`, ook genest;
- `ifdef`, `ifndef` wordt omgezet `in if isdef(...)` of `! isdef(..)` en kan dus ook worden gebruikt binnen Booleaanse uitdrukkingen;
- `include <string expression> | exit`
- `message | error`
- `code` (om regels code te maken);
- `macro/endmacro` met passende parameterlabeling;
- een bijkomende eis is dat instructies van bestaande preprocessoren kunnen worden gemengd met die van SysGen zonder elkaar te storen.

Het SysGen-programma kan ook worden gedownload van de opgegeven webpagina [1]. SysGen is geschreven in Java (versie 12) en vereist een corresponderende Java-installatie om te kunnen draaien.

## Programmeren met Metronom

Om de gebruiker de vervelende taak van het doorwerken van de broncode van het besturingssysteem te besparen, is het hele besturingssysteem zo opgebouwd dat het automatisch wordt gegenereerd. Dat betekent dat de gebruiker alleen het definitiebestand en – indien nodig – de door de gebruiker geprogrammeerde interruptroutines hoeft in te vullen; waar ze thuishoren en hoe ze worden gekoppeld wordt automatisch door het generatorproces verzorgd.
In zijn basisvorm bestaat een gebruikerssysteem uit de in **figuur 3** geschetste onderdelen.
De interrupttabel en de kernel worden altijd als één geheel opgenomen in het resulterende algemene programma. In het geval van de device handlers en de libraries daarentegen worden alleen de werkelijk benodigde delen opgenomen.
Ter illustratie van het generatorproces is het generatorscript van een van mijn eigen projecten te zien in **listing 1**.

210719-03

## Vragen of opmerkingen?

Hebt u technische vragen of opmerkingen naar aanleiding van dit artikel? Stuur een e-mail naar de auteur via profos@rspd.ch of naar de redactie van Elektor via redactie@elektor.com.

## Gerelateerde producten

- **A. He and L. He, *Embedded Operating System* (SKU 19228).**
  www.elektor.com/19228
- **A. He and L. He, *Embedded Operating System* (e-boek, SKU 19214).**
  www.elektor.com/19214

## WEBLINKS

[1] Generatorbestanden voor Metronoom, generatorprogramma SysGen, documentatie: http://www.elektormagazine.com/labs/metronom-a-real-time-operating-system-for-avr-processors

**Listing 1**

```
; *********************************************************
; Master Definition
; *********************************************************
; Stand: 03.05.2022
;
; This file contains all informations required to generate your user system for
; AVR processors.
; It consists of three parts:
;
; 1. Definitions
;    A bunch of variable definitions defining which functionalities to include.
;    This part must be edited by the user.
;
; 2. An $include statement for the actual generation of the operating system.
;    DO NOT MODIFY THIS STATEMENT!
;
; 3. The $include statement(s) adding the user program(s).
;    This part must be edited by the user.
;

; *********************************************************
; PART 1: DEFINITIONS
;
; This script is valid for ATmega8, ATmega328/P and ATtiny25/45/85 processors.
; If you want to use it for any other processors feel free to adapt it accordingly.

$define processor = "ATmega8"

; Remove the ; in front of the $set directive if you want to use the EEPROM
; $set _GEEPROM=1
; if you want to write your own routines to write to the EEPROM use the following
; definition:
; $set _GEEPROM=2
; Enabling this definition will insert an appropriate JMP instruction to your
; interrupt service routine e_rdy_isr in the InterruptHandlers.asm file
; Remove the ; in front of the $set directive if
; ... you want to output serial data via the USART, or
; ... you want exception messages to be sent outside via the USART
; $set _GUSART=1
; if you want to write your own routines to use the USART
; use the following definition instead
; $set _GUSART=2
; Enabling this definition will enable the interrupt service routines usart_udre_isr,
; usart_rxc_isr and usart_txc_isr in the InterruptHandlers.asm file.

; ---------------------------------------------------------
; Define the division ratios of the time intervals for cyclic tasks
; The definition shown here is the standard preset for 1 : 10 : 100 : 1000 ms
; The first ratio being 0 ends the divider chain.
.equ _KRATIO1 = 10◦◦◦◦◦◦◦◦◦◦◦ ; 1 -> 10ms
.equ _KRATIO2 = 10◦◦◦◦◦◦◦◦◦◦◦ ; 10 -> 100ms
.equ _KRATIO3 = 10◦◦◦◦◦◦◦◦◦◦◦ ; 100ms -> 1s
.equ _KRATIO4 = 0◦◦◦◦◦◦◦◦◦◦◦◦ ; end of divider chain
.equ _KRATIO5 = 0
.equ _KRATIO6 = 0
.equ _KRATIO7 = 0
; NOTE: Do not remove "superfluous" .EQU statements but set them to 0 if not used!

; ---------------------------------------------------------
; Define the constants used for generation of the 1ms timer interrupt
; IMPORTANT: The following definitions depend on the processor being used
; and the frequency of the master clock
```

```
$if (processor == "ATmega8")
; The definitions below are based on a system frequency of 12.288 MHz (crystal)
; This frequency has been chosen in order to use the crystal also for USART@9600 Bd
;
; set prescaler for counter0 to divide by 256, yields 48kHz counting freq for Counter0
.equ _KTCCR0B_SETUP = 4
; Counter0 should divide by 48 in order to produce interrupts every 1ms;
; since counter0 produces an interrupt only at overflow we must preset
; with (256-48) - 1 = 207.
$code ".equ _KTCNT0_SETUP = " + (256 - 48) - 1

$elif ... similar for other processors
;
$endif
;
; ------------------------------------------------------------
; Define the characteristics of USART transmission
; (if you don't use the USART just neglect these definitions):
$set fOSC = 12288000
$set baud_rate = 9600
$code ".equ _KUBRR_SETUP = " + (fOSC / (16 *baudrate) - 1)

; parity: 0 = Disabled,
; (1 = Reserved), 2 = Enable Even, 3 = Enable Odd
.equ _KPARITY = 0

; stop bits: 0 = 1 stop bit, 1 = 2 stop bits
.equ _KSTOP_BITS = 1

; data bits transferred: 0 = 5-bits, 1 = 6-bits, 2 = 7-bits, 3 = 8-bits, 7 = 9-bits
.equ _KDATA_BITS = 3
;
; ------------------------------------------------------------
; Connect a user defined interrupt handler (except RESET and Timer0)
; by removing the ; in front of the appropriate $set directive;
; don't change any names but just let the $set statement as is

; Interrupts for ATmega8
; $set _kext_int0 = 1    ; IRQ0 handler
$set _kext_int1 = 1      ; IRQ1 handler/initializer is supplied by user
; $set _ktim2_cmp = 1    ; Timer 2 Compare Handler
; $set _ktim2_ovf = 1    ; Timer 2 Overflow Handler
; $set _ktim1_capt = 1   ; Timer 1 Capture Handler
;
; etc. etc. etc.

;
; *********************************************************
; PART 2: GENERATING THE OPERATING SYSTEM
;
.LISTMAC
;
$include lib_path + "\GenerateOS.asm"
;
;
; *********************************************************
; PART 3: ADD THE USER PROGRAM
;
$include user_path + "\MyApplication.asm"
;
$exit
```