le système d'exploitation temps réel Metronom

un RTOS pour processeurs AVR

Dieter Profos, Dr. sc. techn. ETH (Suisse)

Pour de nombreuses tâches - comme le traitement de signaux continus - les microcontrôleurs doivent effectuer des opérations dans des intervalles de temps précis. Le système d'exploitation en temps réel présenté ici est (également) adapté aux contrôleurs AVR disposant d'un espace mémoire limité. Vous devez accepter certaines limitations, dont notamment la programmation en assembleur, qui constitue néanmoins un bon compromis pour les projets où la vitesse et la capacité en temps réel sont importantes.

Mais pourquoi un autre système d'exploitation ?

Avec l'apparition de petits et micro processeurs et contrôleurs, certains processus, pour lesquels l'utilisation d'un *vrai* ordinateur n'aurait jamais été nécessaire auparavant, sont devenus automatisables. Ces microcontrôleurs n'ont pas besoin de contrôler les périphériques (clavier, souris, écran, disque, etc.), de sorte que les systèmes d'exploitation peuvent être limités à l'essentiel : organiser le traitement des programmes utilisateur.

La plupart des systèmes d'exploitation sont conçus pour exécuter le plus de programmes possible de la manière la plus efficace possible et (côté utilisateur) simultanément. Les choses sont toutefois différentes lorsqu'il s'agit de traiter des signaux continus et limités dans le temps : Cela nécessite des fonctions qui s'exécutent pendant des intervalles précis. Par exemple, la fonction `delay()` d'Arduino n'est plus suffisamment précise dans ce cas, car elle ne génère que des temps d'attente, mais ne prend pas en compte les temps d'exécution nécessaires au traitement, qui sont explicites avec des temps d'échantillonnage de 1 ms ou même plus courts.

Les deux problèmes suivants doivent donc être résolus :

- Certaines tâches doivent être exécutées exactement à des moments prédéfinis, d'autres seulement lorsqu'il leur reste du temps.
- Chaque tâche interruptible nécessite sa propre pile pour mettre en mémoire tampon le contenu des registres. Cependant, avec les microcontrôleurs, l'espace mémoire est assez limité : par exemple, 750 octets pour l'ATtiny25 ou 1 K pour l'ATmega8.

Le système d'exploitation Metronom présenté ici est disponible en téléchargement sur le site web d'Elektor [1] en tant que logiciel libre sous la licence BSD-2 ; une version via GitHub est également prévue dans les mois à venir.

Tâches cycliques

Metronom est conçu précisément pour l'exécution de ce que l'on appelle des tâches cycliques pendant des intervalles de temps prédéterminés (jusqu'à 8 durées de cycle différentes). Une seule tâche par cycle ; si différentes opérations au contenu indépendant doivent être exécutées dans le même cycle, elles doivent être combinées dans la même tâche.

Les temps de cycle sont générés comme suit :

- La période du cycle de base (par ex. 1 ms) est établie avec

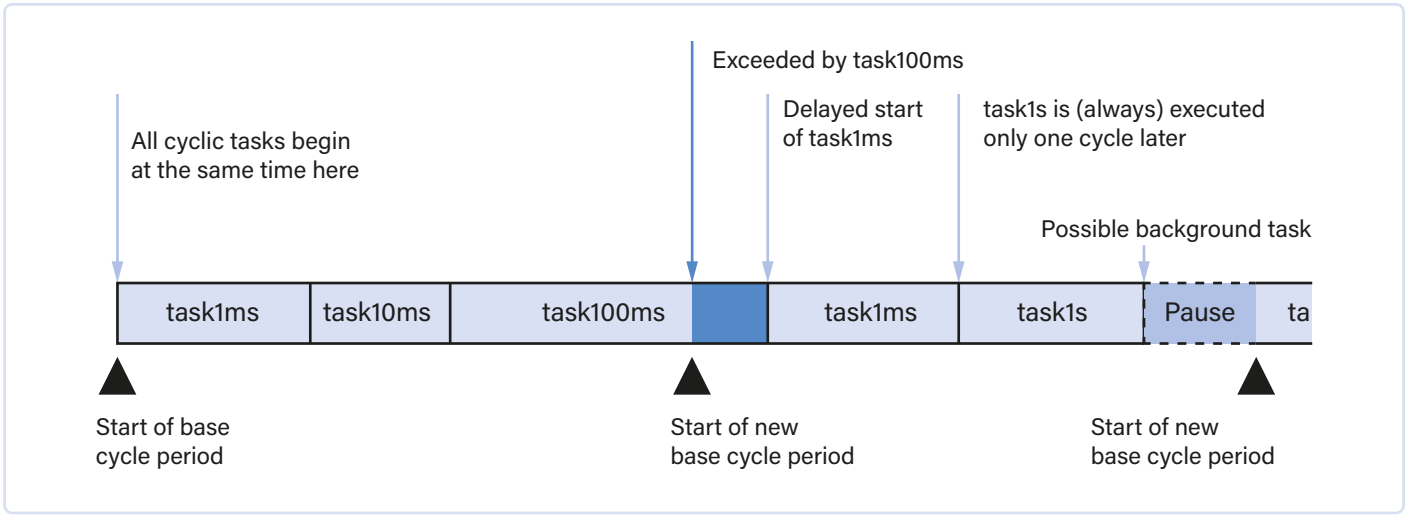

Figure 1. Séquence de plusieurs tâches cycliques (cas limite).

le matériel du processeur (quartz ou oscillateur RC interne, compteur logiciel contrôlé par le matériel et les interruptions), et
- les autres durées de cycle sont générées par une succession de compteurs, de sorte que chaque durée de cycle est un multiple de la précédente (par exemple, si le paramètre par défaut est 1 ms → 10 ms → 100 ms → 1 s).

Le système d'exploitation présenté ici a une caractéristique importante : les tâches cycliques ne peuvent pas s'interrompre (non préemptives). D'une part, cela garantit que le *timing* de ces tâches est aussi précis que possible, et d'autre part, le processeur ne gaspille pas du *temps improductif* à changer de tâche. Et comme chaque tâche cyclique – une fois lancée – s'exécute complètement sans interruption (sauf par des interruptions) avant que la tâche cyclique suivante ne soit lancée, toutes les tâches cycliques peuvent utiliser la même pile.
Mais que se passe-t-il si l'exécution d'une tâche dure plus que la période du cycle de base (ce qui doit être évité par le développeur) ou si plusieurs tâches cycliques ont été lancées dans le même cycle, où la somme des temps d'exécution dépasse la durée du cycle de base (ce qui est tout à fait justifié) ? Ici une autre caractéristique de Metronom intervient : les tâches cycliques ont différentes priorités : la tâche avec le temps de cycle le plus court a la plus haute priorité, la *deuxième* tâche la plus rapide a la deuxième priorité, et ainsi de suite. Si toutes les tâches cycliques lancées en même temps ne peuvent pas être achevées dans le temps de cycle de base, la tâche en cours d'exécution est poursuivie jusqu'à sa fin après l'écoulement de la durée du cycle de base – mais alors la tâche *la plus rapide* la plus prioritaire est exécutée en premier, et ce n'est qu'ensuite que les autres tâches cycliques lancées précédemment sont exécutées à nouveau.
Exemple : l'implémentation des temps de cycle entraîne le lancement simultané de toutes les tâches cycliques toutes les secondes. Ce qui se passe dans ce cas est illustré à la **figure 1**.
Cela signifie que chaque tâche cyclique ne doit pas dépasser le temps de cycle le plus court – c'est-à-dire 1 ms comme limite supérieure absolue dans notre exemple. Cela correspond à environ 6 000 instructions pour un ATtiny fonctionnant à 8 MHz et à environ 14 000 instructions pour un ATmega à 16 MHz (le reste des instructions est utilisé – en moyenne – par le système d'exploitation lui-même et pour la gestion des interruptions).

Tâches d'arrière-plan

Toutefois, certaines opérations prennent plus de temps à cause de leur nature :
- L'accès en écriture à l'EEPROM, par exemple, prend quelques millisecondes (typiquement environ 3,3 ms), c'est-à-dire un temps excessivement long pour un cycle de base de 1 ms.
- Transmettre du texte à 9 600 Bd est impossible dans un cycle de base de 1 ms car même la transmission d'un seul caractère prend déjà plus de 1 ms.
- Lorsque des calculs plus longs (par exemple, des opérations arithmétiques émulées) sont nécessaires ou que des chaînes de caractères doivent être traitées, cela prend souvent trop de temps au cours d'une tâche cyclique et bloque donc les processus soumis à des contraintes de temps.

Cela signifie qu'il faut encore trouver un moyen de réserver ces processus aux tâches interruptibles. Deux procédés combinés sont utilisés à cet effet :
- Utilisation d'interruptions au lieu de l'attente active : cela permet *d'affecter* l'attente de la fin d'une opération (par exemple, la transmission d'un caractère) au matériel ; cette méthode est utilisée pour les opérations contrôlées par des interruptions. Cela résout les problèmes de transmission d'un seul caractère ou d'écriture d'une seule valeur dans l'EEPROM, mais pas l'attente de la fin de l'opération globale (par exemple, la transmission d'un texte entier).
- Mise en œuvre des tâches d'arrière-plan : une tâche d'arrière-plan s'exécute uniquement pendant les phases inoccupées par les tâches cycliques. En outre, elle peut être interrompue à tout moment, de sorte qu'elle n'interfère pas avec l'exécution ponctuelle des tâches cycliques.

Cependant, une fois qu'une tâche d'arrière-plan est en cours d'exécution, elle ne peut pas être interrompue par d'autres tâches similaires. Ainsi, une seule tâche d'arrière-plan est traitée à la fois, et si elle est suspendue, le traitement des tâches d'arrière-plan l'est également. Bien que cela ralentisse le traitement des tâches d'arrière-plan, cela signifie qu'un seul emplacement de pile doit être réservé pour toutes les tâches d'arrière-plan.

Les tâches d'arrière-plan se caractérisent par :
- Une tâche d'arrière-plan peut être interrompue à tout moment en faveur de tâches cycliques, mais pas en faveur d'une autre tâche similaire.
- L'exécution d'une tâche d'arrière-plan est déclenchée par un appel au répartiteur.

- Les tâches d'arrière-plan sont exécutées l'une après l'autre dans l'ordre où elles ont été lancées.
- Les tâches d'arrière-plan peuvent attendre des événements (`WAIT_EVENT`), qui sont déclenchés, par exemple, par des processus contrôlés par des interruptions.
- Les tâches d'arrière-plan peuvent également être mises en attente pendant des durées prédéfinies (`DELAY`).
- Chaque tâche d'arrière-plan peut recevoir un message de départ de 3 mots de 16 bits, qui peut être utilisé pour spécifier son objectif (un 4e mot est réservé pour l'adresse de départ de la tâche).
- Un nombre quelconque de routines de tâches d'arrière-plan existe dans le programme utilisateur ; toutefois, un maximum de 8 peuvent être lancées simultanément.

La coordination des tâches entre elles (à *quel moment l'exécution de telle ou telle tâche est-elle autorisée* ?) est gérée par ce que l'on appelle le répartiteur. Il exécute tous les « processus administratifs », tels que le démarrage des tâches, la sauvegarde/restauration des registres du processeur, ou la désactivation/activation des interruptions.

Exceptions

Comme les microcontrôleurs ne disposent généralement pas de périphériques orientés texte, le débogage est très compliqué, notamment pour les fonctions dépendant du temps, car les points d'arrêt ou autres perturbent complètement le comportement temporel. Par conséquent, le noyau du système d'exploitation fournit un mécanisme simplifié pour le traitement des exceptions, qui est constitué de deux étapes :

- Une zone `try-catch` globale détecte toutes les exceptions (exceptions/erreurs) survenant dans le noyau et dans les émulations arithmétiques. Les données spécifiques à l'exception peuvent être stockées dans l'EEPROM et/ou sorties via USART ; ensuite, le système d'exploitation effectue un *RESET* total du système (y compris le `user-reset`). Cette zone d'exception est toujours active.
- En outre, il est possible d'utiliser une zone `try-catch` orientée application, qui ne traite que le programme utilisateur en cours. Le traitement de ces exceptions se fait initialement ainsi : les données d'exception sont stockées et/ou sorties via USART ; ensuite, une routine de *redémarrage de l'application* à spécifier par l'utilisateur est exécutée (sous-routine `user_restart`).

Gestion des interruptions

Les interruptions sont traitées de quatre manières différentes :

- L'interruption de réinitialisation est utilisée par le système d'exploitation et n'est pas directement accessible par l'utilisateur. Cependant, comme l'utilisateur a également besoin de cette interruption pour initialiser ses processus, le système d'exploitation appelle la sous-routine `user_init` après sa propre initialisation, qu'il peut utiliser avec son code d'initialisation spécifique à l'application.
- Timer/counter0 est utilisé pour la génération de l'horloge de base pour tous les processus cycliques ; il n'est donc pas accessible par l'utilisateur.
- Pour l'utilisation de l'EEPROM et de l'USART, le système d'exploitation propose des blocs pilotes prêts à l'emploi, qui peuvent être intégrés lors de la génération du système d'exploitation (voir ci-dessous). Cependant, l'utilisateur peut, de plus, associer ses propres routines de service à ces interruptions ou simplement les laisser ouvertes lorsqu'elles ne sont pas utilisées.
- Toutes les autres interruptions sont directement à disposition de l'utilisateur. Pour chaque interruption, une routine de service d'interruption ainsi qu'une routine d'initialisation d'interruption doivent être spécifiées ; si plus d'une interruption appartient à un périphérique (par exemple, des timers ou USART), une routine d'initialisation partagée est suffisante. Pour cela, l'utilisateur active les paramètres correspondants dans le fichier de génération et insère le contenu des routines d'initialisation et de service correspondantes dans son programme utilisateur.
- Les interruptions non utilisées sont automatiquement « interceptées » par le système d'exploitation.

Environnement de programmation

Pour des raisons d'efficacité, Metronom est écrit en assembleur AVR (Atmel/Microchip) et implique donc que les programmes utilisateurs soient également écrits en assembleur ; aucune interface pour le langage C n'a été implémentée. Cependant, il existe une bibliothèque avec de nombreux sous-programmes, pour l'arithmétique 8 bits ainsi que l'arithmétique 16 bits (4 opérations arithmétiques de base) entre autres ; une bibliothèque de fractionnement 16 bits est en cours de préparation. Pour faciliter la programmation, tous les appels du système d'exploitation sont disponibles sous forme de macros. Pour éviter les conflits de nommage, la convention d'affectations de noms suivante s'applique : toutes les variables et les cibles de saut dans le système d'exploitation et les bibliothèques commencent par un trait de soulignement (« _ »). Par conséquent, tous les noms dans le programme utilisateur doivent uniquement commencer par des lettres. Les caractères autres que les lettres, les chiffres et le trait de soulignement ne sont pas autorisés. La structure globale du Metronom et le programme utilisateur correspondant sont illustrés dans la **figure 2**.

Appels du système d'exploitation

Pour la liste complète des appels du système d'exploitation, veuillez vous reporter aux références en fin d'article ; nous n'en donnons ici qu'un aperçu :

Macros pour le traitement des exceptions

- `KKTHROW` lance une exception à l'échelle du système, c'est-à-dire qu'après avoir sauvegardé/sorti les informations relatives à l'exception, le système entier est redémarré.
- `KTHROW` lance une exception limitée au programme utilisateur, c'est-à-dire qu'après la sauvegarde/la sortie des informations relatives à l'exception, seule la sous-routine utilisateur `user_restart` est exécutée ; ensuite, les tâches cycliques sont relancées.

Macros pour l'utilisation des tâches d'arrière-plan

- `_KSTART_BTASK` lance une tâche d'arrière-plan.
- `_KDELAY` met en veille la tâche d'arrière-plan appelante pendant n (0 à 65 535) ms.
- `_KWAIT` met en veille la tâche d'arrière-plan appelante, qui peut être reprise avec ...
- `_KCONTINUE`.

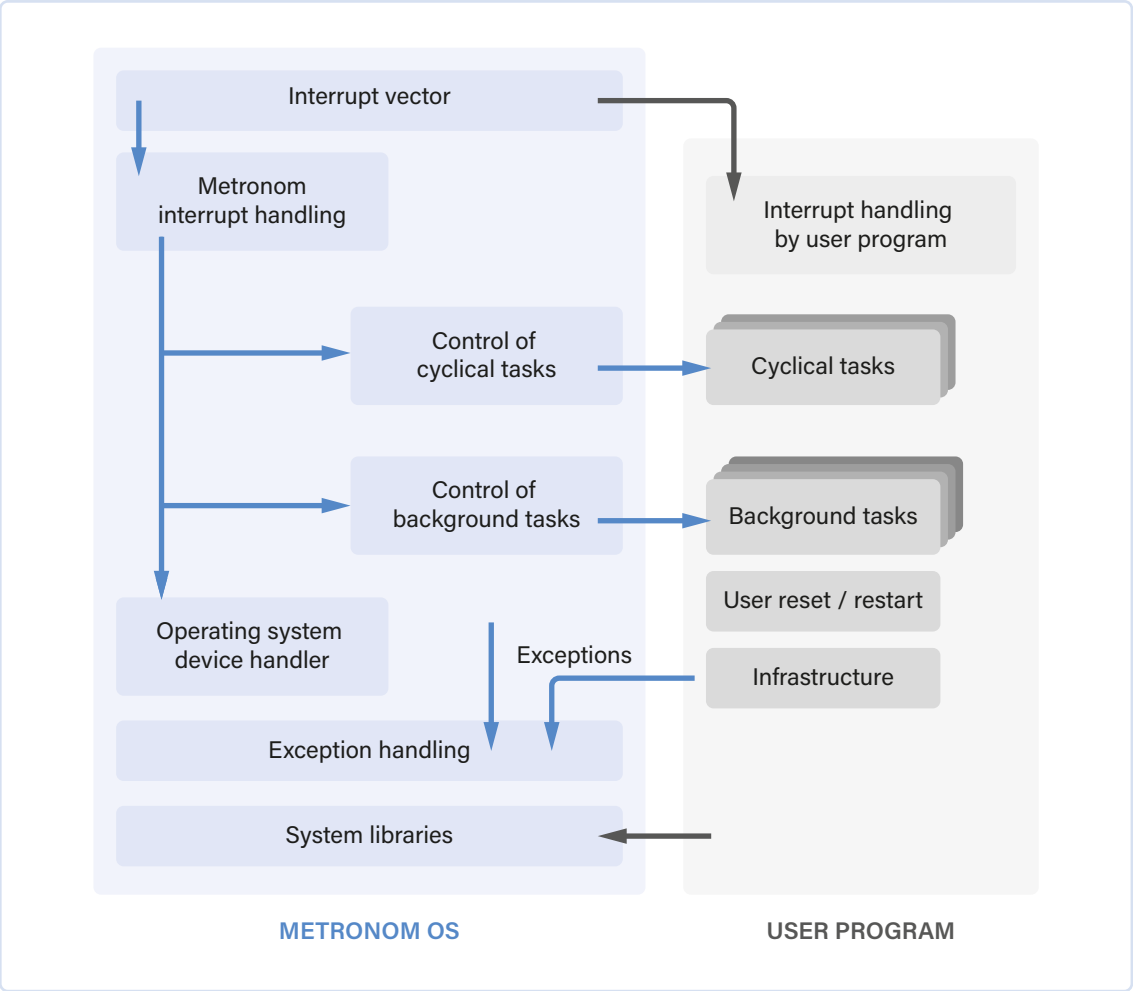

Figure 2. Structure générale du Metronom et du programme utilisateur.

Macros pour l'arithmétique 8 bits et 16 bits

En général, pour les opérations arithmétiques de toutes sortes, les registres r25:r24 sont utilisés comme accumulateur et r23:r22 comme mémoire pour le second opérande (si nécessaire). À cette fin, il existe plus de 20 fonctions différentes, telles que `_mul8u8` pour une multiplication 8×8 bits ou `_abs16` pour une valeur absolue de 16 bits. En outre, il existe de nombreux pseudo-codes de chargement et de sauvegarde, comme `_ld16` (chargement d'un nombre de 16 bits dans l'accumulateur).

Macros pour l'utilisation de l'EEPROM

- `_KWRITE_TO_EEPROM` pour écrire dans l'EEPROM
- `_KREAD_FROM_EEPROM` pour la lecture de l'EEPROM

Macros pour l'utilisation de l'USART

- `_KWRITE_TO_LCD` est un pilote USART spécifique pour un écran LCD 2×16, qui ajoute les caractères de contrôle nécessaires au texte à afficher.
- `_KREAD_FROM_USART` (pas encore implémenté).

Générateur de système SysGen

Pour générer un système (c.-à-d, le code complet), un générateur de système dédié SysGen, qui fait également partie du paquet global, est utilisé. SysGen n'est pas limité à Metronom, mais peut également être utilisé pour des tâches de génération générales.

Vous pouvez vous demander pourquoi un générateur de système distinct a été développé, étant donné qu'il existe une grande variété de préprocesseurs et de générateurs de macros. Pour la génération du système d'exploitation Metronom, les fonctionnalités des préprocesseurs d'Atmel Studio ainsi que du C standard ne sont pas suffisantes. En particulier, comme le préprocesseur ne prend pas en charge l'arithmétique des chaînes, il est impossible de spécifier un « répertoire par défaut » ou un « répertoire de bibliothèque » et d'y sélectionner les fichiers qu'il contient. En faisant une recherche sur Stack Overflow, j'ai constaté que d'autres personnes ont le même problème que moi, mais aucun des préprocesseurs existants ne peut le résoudre.

Le préprocesseur d'Atmel Studio (ainsi que le préprocesseur GNU) offre les fonctions suivantes pour rassembler les fichiers requis :

- `define / set =`
- `if … elif … else … endif`, également intégré
- `ifdef, ifndef`
- `include | exit`

Il manque les fonctionnalités suivantes :

- `<path>` ne peut être passée que sous la forme d'une chaîne fixe, mais une expression de (n'importe quel nombre) chaînes partielles, à la fois des variables et des constantes de chaîne, serait nécessaire.
- `define` et `set` ne peuvent affecter que des valeurs numériques, pas de chaînes de caractères, pas de concaténation de chaînes de caractères, et pas non plus d'expressions logiques.
- Pour la declaration (unique) de programmes de bibliothèque, les possibilités de macros offertes par AVRASM ou le préprocesseur en C ne sont pas suffisantes ; comme les macros d'AVRASM

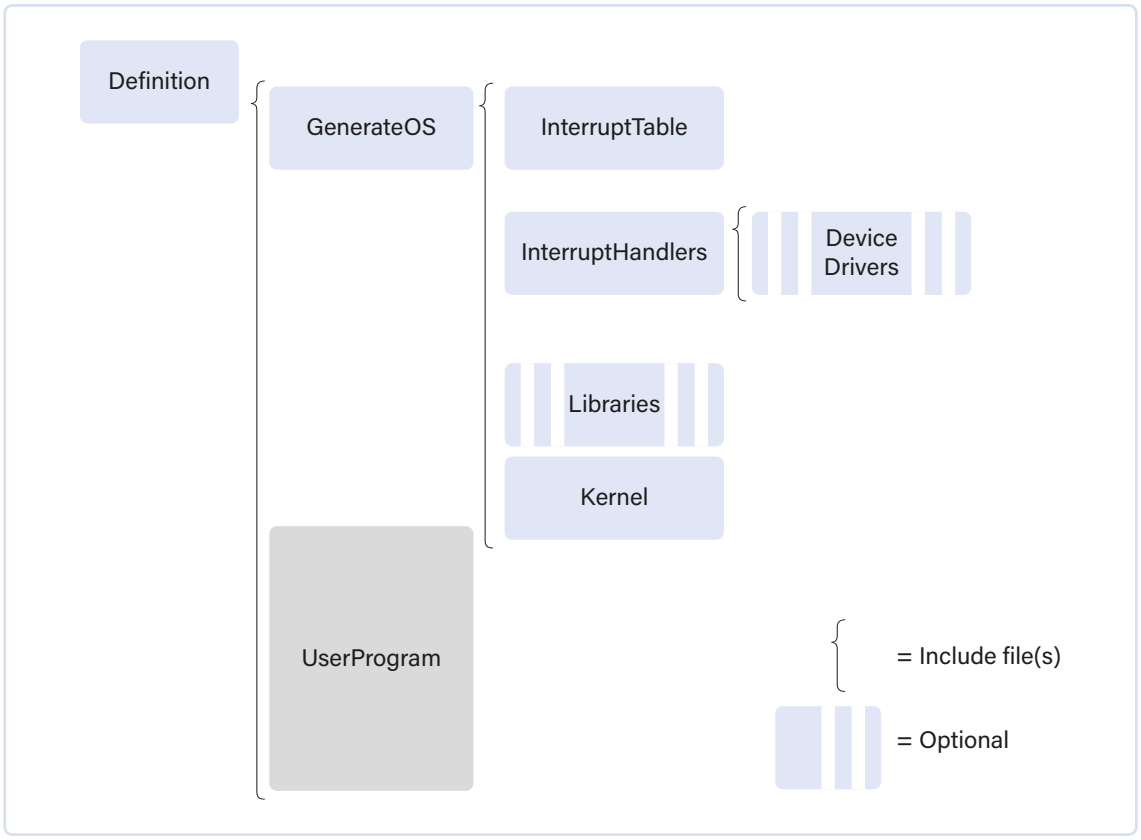

Figure 3. Structure de génération des systèmes Metronom.

ne peuvent pas contenir d'instructions *include*, la déclaration automatique de routines d'émulation, par exemple, est impossible.

Cela donne lieu aux fonctions suivantes :

- `define / set = | |`
- `if … elif … else … endif`, également intégré
- `ifdef, ifndef` est converti en `if isdef(..)` ou `! isdef(..)` et peut donc également être utilisé dans des expressions booléennes.
- `include | exit`
- `message | error`
- `code` (pour créer des lignes de code)
- `macro/endmacro` avec un étiquetage approprié des paramètres
- Il faut également que les instructions des préprocesseurs existants puissent être combinées avec celles de SysGen sans interférer les unes avec les autres.

Vous pouvez également télécharger le programme SysGen sur [1]. SysGen est écrit en Java (version 12) et nécessite une installation Java correspondante pour fonctionner.

Programmer avec Metronom

Pour vous faciliter la tâche, l'ensemble du système d'exploitation est structuré pour être automatiquement généré. Cela signifie que l'utilisateur n'a qu'à remplir le fichier de définition et – si nécessaire – les routines d'interruption programmées par lui-même ; leur emplacement et la manière dont elles sont connectées sont assurés automatiquement par le processus de génération.

Dans sa forme de base, un système utilisateur est composé des éléments illustrés à la **figure 3**.

La table d'interruption et le noyau sont toujours incorporés ensemble dans le programme global résultant. En revanche, dans le cas des gestionnaires de périphériques et des bibliothèques, seules les parties qui sont réellement nécessaires sont incorporées.

Pour illustrer le processus de génération, le script de génération d'un de mes propres projets est présenté dans le **listage 1**.

210719-04

Des questions, des commentaires ?

Envoyez un courriel à l'auteur (profos@rspd.ch) ou contactez Elektor (redaction@elektor.fr).

Produits

- ***Livre en anglais « Embedded Operating System », A. He and L. He*** **(SKU 19228)**
  Version papier
  https://elektor.fr/19228
- Version numérique
  https://elektor.fr/19214

LIENS

[1] Generation files for Metronome, generation program SysGen, documentation : http://www.elektormagazine.fr/labs/metronom-a-real-time-operating-system-for-avr-processors

Listage 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
```