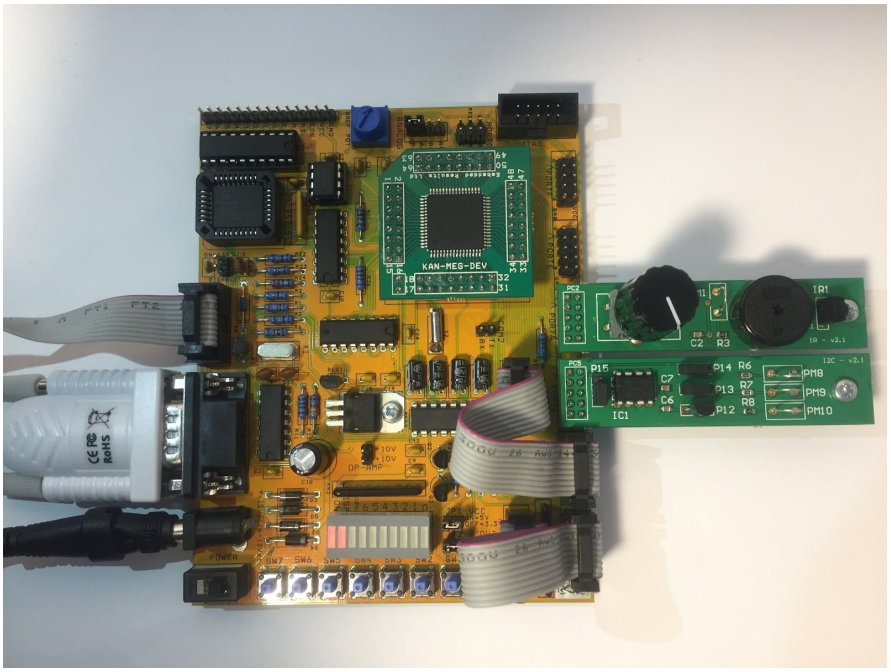
SECTION MICROTECHNIQUE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Rapport de Projet de fin de Semestre

Tratagua



Numéro du groupe	L16 (Diane Marquette & Luc Reveyron)
Assistant responsable	
Enseignant	Schmid Alexandre
Semestre	BA 4
Cours	Microcontrôleurs
Date	30/05/2018
Signatures	

TABLE DES MATIÈRES

1.1. INTRODUCTION

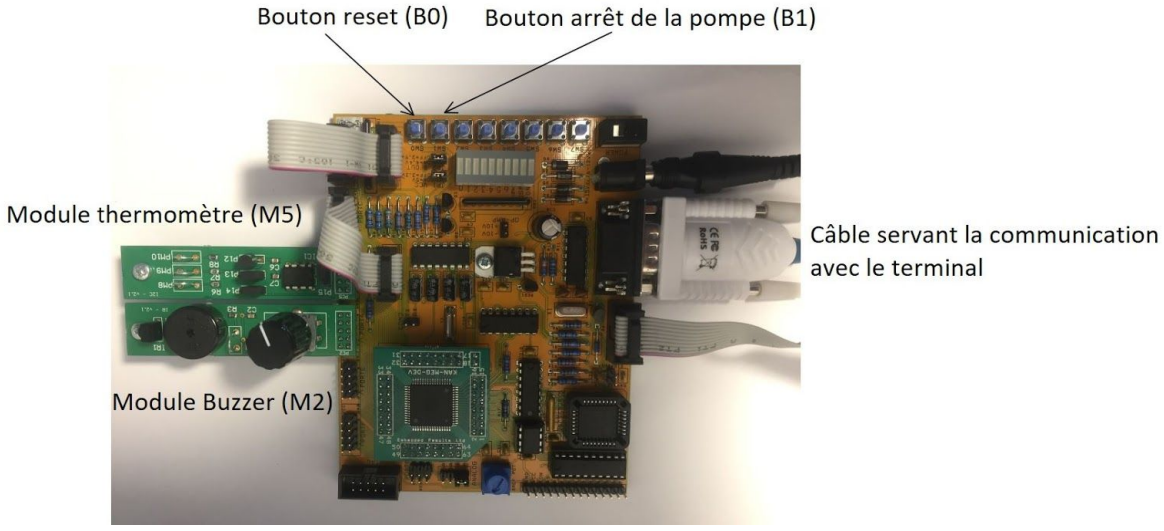
Le programme Tratagua a été développé pour réaliser l'interface utilisateur d'un système de traitement des eaux usées, écologique et adapté à des bâtiments non-reliés au tout-à-l'égoût.

Ce dispositif, déjà utilisé en République dominicaine, comporte notamment une pompe de relevage. Il est essentiel que l'utilisateur soit avisé dans les plus brefs délais en cas de panne, en recevant un message, par exemple. La température est également un paramètre important à surveiller.

1.2. MANUEL D'UTILISATION

Où et que connecter à la carte STK300?

Branchez le buzzer (module M2) sur le port A et le thermomètre (module M5) sur le port B. Reliez les boutons au port D. Connectez l'ordinateur et la STK300 par le câble d'interface pour la communication UART.



Interagir avec la STK300

- Pour modifier l'intervalle de temps entre deux notifications, entrez l'une des quatre lettres suivantes dans le terminal.

LETTRE	SIGNIFICATION	INTERVALLE DE TEMPS ENTRE DEUX MESSAGES
s	second	5 secondes
f	five	5 minutes

t	twelve	12 heures
d	day	24 heures

- Pour demander à tout moment l'état du système, tapez "u".

-Pour simuler l'arrêt de la pompe, appuyez sur le bouton 1. Une message urgent est alors envoyé (figure 2) et un son est émis.

Exemple de notification

Exemple de message d'alerte

-Pour effectuer un RESET, appuyez sur le bouton 0.

1.3. FONCTIONNEMENT DU PROGRAMME

1.3.1. Fonctionnement général (utilisateur)

Comme détaillé dans le manuel d'utilisation, l'utilisateur peut:

- ➔ demander une update concernant l'état de la pompe et la température (terminal UART);
- ➔ modifier l'intervalle de temps entre l'envoi de deux messages (terminal UART);
- ➔ faire un reset en appuyant sur le bouton 0.

La pompe est modélisée par un bouton. On considère:

- qu'elle fonctionne quand le bouton 1 est relâché;
- qu'elle est en panne quand le bouton 1 est appuyé.

Le son de l'alarme est généré le buzzer piézo-électrique du module M2. La température est mesurée par le thermomètre du module M5 fourni avec le kit.

Notre programme est composé de quatre interruptions:

- ➢ INT0 - effectue le reset;
- ➢ INT1 - appelée en cas de panne de la pompe (message urgent + alarme);
- ➢ ovf0 - appelée lors de l'overflow du timer 0 (qui permet de compter le temps écoulé depuis la dernière notification programmée);
- ➢ uart_rcx - appelée lorsque l'utilisateur entre un caractère dans le terminal UART.

Si notre programme n'est pas en train d'exécuter une des routines de service des interruptions, le microcontrôleur est en veille (*sleep mode*).

1.3.2. Choix des registres

Les registres ont été précisément attribués (comme indiqué dans le tableau ci-dessous) pour éviter toute corruption de données.

REGISTRES	UTILISATION
r21	Compteur pour une boucle servant à faire sonner le buzzer
r22	Compteur indiquant le nombre de secondes
r23	Compteur indiquant le nombre de minutes
r24	Compteur indiquant le nombre d'heures
r25	Indique l'origine du message lorsque l'on appelle la sous-routine display - 0 pour l'overflow du timer0 - 1 pour l'interruption INT1 enclenchée par une panne de la pompe
r28	Sauvegarde l'état de la pompe (0 pour ON et 1 pour OFF)
r29	Sauvegarde le mode du timer (seconde, minute, heure ou jour)
a0 (r18), a1 (r19)	Registre temporaire de liaison entre les sous-routines de wire1 et de l'UART

Les registres r26, r27, r30 et r31 (associés aux pointeurs x et z) sont utilisés par les sous-routines des fichiers "printf.asm" et "wire1.asm". Nous n'avons donc pas pu les mettre à profit pour sauvegarder des données. C'est pourquoi nous avons choisi r28 et r29 (correspondant au pointeur y). Il s'agit des seuls registres de ce type, non altérés par des routines ou macros des librairies standards.

1.3.3. Macros, sous-routines et librairies

Pour développer ce projet, nous avons réutilisé plusieurs bibliothèques vues en cours comme celles des macros, de l'uart, de wire1 et de printf. Nous avons toutefois été amenés à modifier le fichier .asm de l'UART pour activer l'interruption "RX Complete".

Nous avons également rédigé une macro et deux sous-routines incluses dans le fichier "personal_library.asm".

NOM	ENTRÉES	SORTIES	MODIFIÉES	BUT
display	a0,a1,a2,r21,r28	-	-	affiche des messages préformatés

temperature	-	a0,a1,a2,c0	a0,a1,a2,c0	lit la température aux bornes du thermomètre
TIMERLIMIT	3 constantes	r22,r23,r24	r22,r23,r24	sauvegarde dans r22 , r23 et r24 des valeurs données en entrée

Nous avons choisi de faire de TIMERLIMIT une macro car il s'agit d'un morceau de code court, composé de seulement quatre instructions. Même si ces quatre lignes sont copiées dans le code assemblé à chaque appel de la macro, cela reste plus intéressant qu'une sous-routine correspondant à un délai supplémentaire de sept cycles.

Par contre, pour les raisons énoncées ci-dessus, le choix du format d'une sous-routine était le plus adapté pour display et temperature.

Le nombre de sous-routines développées pour ce projet est donc inférieur à celui initialement prévu dans le cahier des charges. En approfondissant le développement du pseudo-code, nous sommes rendus compte qu'il était en effet inutile de faire du buzzer et de la lecture de la pompe, des sous-routines, car chacun de ces morceaux n'est utilisé qu'à un seul endroit dans le programme.

1.3.4. Sleep mode

Notre main contient exclusivement un sleep mode. La consommation énergétique du microcontrôleur est donc réduite en dehors de l'exécution des routines de service des interruptions.

Il est vrai que la consommation n'est pas un paramètre critique pour ce projet (car dans le cadre de ce système de traitement des eaux usées, le microcontrôleur sera toujours branché au secteur). Toutefois, minimiser l'énergie de fonctionnement du dispositif reste toujours un atout, notamment d'un point de vue écologique.

1.3.5. Détails du fonctionnement: interruptions

Quatre interruptions sont utilisées dans ce microprojet.

INT0 (reset)

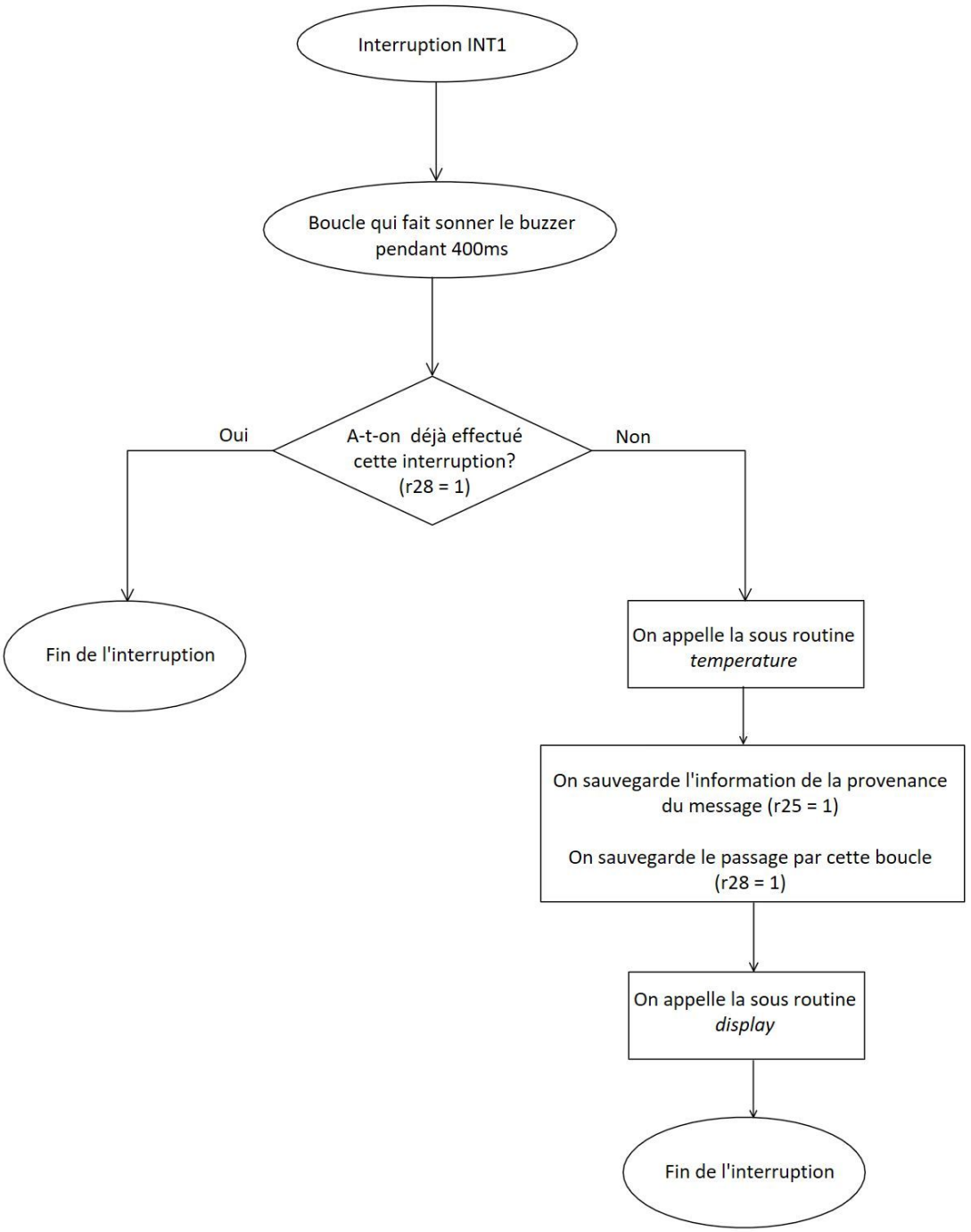
Appuyer sur le bouton 0 entraîne un reset. Ce dernier autorise notamment l'interruption INT0 et INT1, configure les pins du port A en sortie pour le buzzer. De plus, il initialise l'UART ainsi que 1-wire et autorise les interruptions liées au TIMER0. Finalement, il remet à zéro les registres servant à sauvegarder des données lors de l'exécution (r22 : compteur de secondes, r25 : l'origine du message, r28 : le type de message et r29 : le mode du timer) et il autorise les interruptions grâce à la commande sei.

INT1 (pompe + buzzer avec compteur)

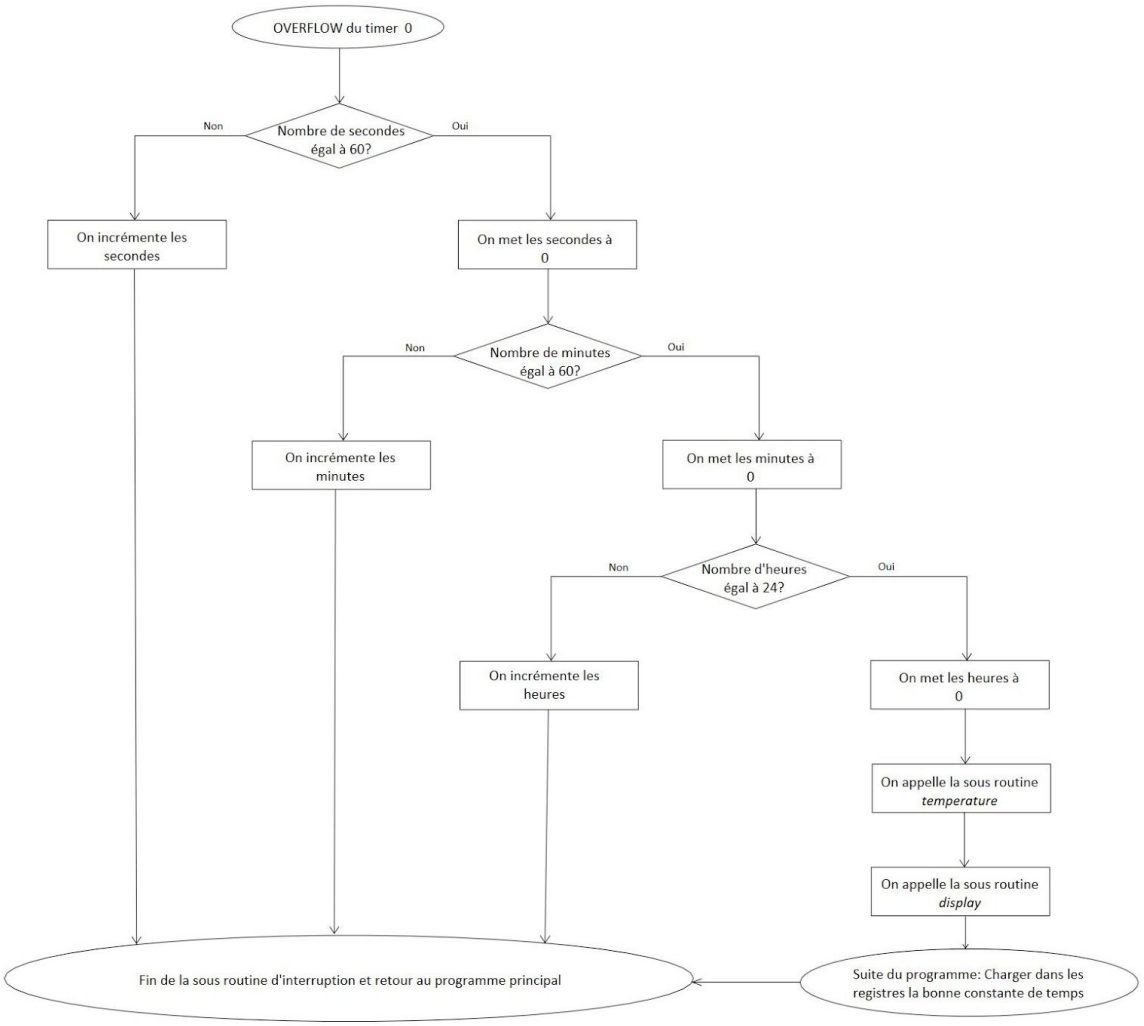
L’interruption 1 est provoquée par le fait d'appuyer sur le bouton 1 (connecté au port D). Cette interruption simule la mise à la terre d’une pompe.

Lorsque cela arrive, le programme émet un son par l'intermédiaire d’un buzzer connecté au port A. Ensuite, le microcontrôleur vérifie si cette interruption a déjà été appelée (ceci permet d'éviter de saturer le terminal de messages d’alerte) et, si ce n’est pas le cas, fait appel à la sous routine display avec le mode message *urgent* (“warning”).

Dans un cas réel (étant équivalent à maintenir le bouton appuyé), l’alarme continue de sonner alors qu’un seul message a été envoyé à l’utilisateur.



ovf0 (overflow du timer)



La routine de service, exécutée lors de l'interruption provoquée par l'overflow du timer 0, incrémente un compteur à trois niveaux.

Ce compteur sert à créer une base de temps allant de la seconde à la journée. Les secondes sont comptées par le registre r22, qui une fois incrémenté jusqu'à 60 passe au registre r23 pour les minutes (incrémenté jusqu'à 60 également) et, enfin, à r24 pour les heures (incrémenté jusqu'à 24). La durée du compteur peut être modifiée lorsque le mode du timer, sauvegardé dans r29, change (0 pour 24h, 1 pour 12h, 2 pour 5 minutes et 3 pour 5 secondes).

Quelque soit l'intervalle de temps, désiré par l'utilisateur, entre deux messages, les limites supérieures des registres r22, r23 et r24, avant leur remise à zéro, restent les mêmes. Ce sont les valeurs initiales à partir desquelles on recommence à compter qui sont modifiées. Ainsi, pour programmer un intervalle de 5 secondes, on précharge systématiquement dans les registres r22, r23 et r24 les valeurs 55, 59 et 23. On trouvera les autres valeurs dans le tableau ci-dessous. Cette opération est faite par la macro TIMERLIMIT.

INTERVALLE DE TEMPS ENTRE 2 MESSAGES	SECONDES (r22)	MINUTES (r23)	HEURES (r24)
5 secondes	55	59	23
5 minutes	0	55	23
12 heures	0	0	12
24 heures	0	0	0

uart_rcx (réception d'une commande de l'utilisateur)

Nous avons autorisé cette interruption en mettant à 1 le bit RXCIE0 du registre UCSR0B. Elle est enclenchée à chaque fois que l'utilisateur entre un caractère dans le terminal UART.

La lettre est convertie en un code binaire. Nous comparons ce dernier à ceux mentionnés dans le tableau ci-dessous pour en déduire la commande envoyée par l'utilisateur.

S'il s'agit d'un intervalle de temps (d, t, f ou s), le registre r29 est modifié en fonction (0, 1, 2 ou 3 respectivement).

S'il s'agit d'une demande d'update (u), une mesure de la température est effectuée et un message est affiché dans le terminal.

LETTRES	CODE BINAIRE ASSOCIÉ
u	0b11110101
d	0b11100100
t	0b11110100
f	0b11100110
s	0b11110011

Les codes binaires associés aux lettres ont tout d'abord été obtenus empiriquement en observant le code reçu par le programme lors d'une communication entre le terminal et la STK300. A posteriori, nous nous sommes rendus compte qu'il s'agissait des codes ASCII dont le bit de poid fort était mis à 1. Par exemple, le code ASCII associé au "u" est 0x75 (en hexadécimal) soit en binaire 0b01110101, en remplaçant le 7ème bit par 1, on obtient le code transmis par l'UART.

1.4 CONCLUSION

Notre système répond donc parfaitement au cahier des charges que nous nous étions défini. Il dépasse même les exigences du professeur, en ce qui concerne le nombre d'interruptions et de périphériques utilisés.

Comme mentionné dans ce rapport, notre projet:

- utilise 4 interruptions;
- interagit avec 3 périphériques (le buzzer, le thermomètre et le terminal UART d'un ordinateur);
- mesure et traite la température, interprète les caractères envoyés par l'utilisateur, et communique à intervalles programmables des messages récapitulatifs;
- possède une interface utilisateur permettant d'entrer des paramètres à l'aide de boutons ou du terminal;
- affiche un résultat sur terminal par UART (RealTerm);
- utilise les macros et librairies introduites dans le cours (telles que "uart.asm", "wire1.asm" ou encore "printf.asm").

Pour implémenter ce microprojet à notre système de traitement des eaux usées, certains aspects doivent être approfondis.

D'une part, la pompe ne sera plus modélisée par un bouton. Nous allons utiliser un optocoupleur. Ce dernier va détecter l'état réel de la pompe, actuellement explicité par une LED sur le tableau électrique.

D'autre part, nous devrons choisir un capteur de température plus adapté, travaillant notamment en milieu aqueux.

Nous comptons relever ces deux défis en poursuivant ce projet cet été et en codant une application rudimentaire pour permettre la réception et l'affichage des messages directement sur le smartphone de l'utilisateur.

1.5 ANNEXES

- le cahier des charges
- les fichiers source assembleur

```
;
; main.asm
; Author : Diane Marquette, Luc Reveyron
;

.include "macros.asm"                   ; include macros definition
.include "definitions.asm"              ; include register/constant definition

;====INTERRUPT TABLE====
.org    0
        jmp reset
        jmp ext_int0
        jmp ext_int1

.org    0x24
        rjmp uart_rcx                   ; UART RX Complete handler


.org    OVF0addr
        rjmp ovf0

;====OTHER INCLUDES====
.org    0x30
.include "uart.asm"                     ; include UART routine
.include "printf.asm"                   ; include formatted printing routines
.include "wire1.asm"                    ; include wire1 routines
.include "personal_library.asm"         ; include the subroutines we have specially
  created for this project
;====INTERRUPT SERVICE ROUTINES====

ext_int0:
        rjmp reset                      ; pressing button 0 induces a reset

ext_int1:                               ; pressing button 1 simulates the reaction to
  a deficient pump  (sound + warning message)

        ldi r21,200                     ; load constant to generate a long enough
          sound

        counter:                        ; loop used to generate a sound
            sbi PORTA,SPEAKER
            WAIT_US 200

            cbi PORTA,SPEAKER
            WAIT_US 200

            dec r21
            cpi r21,0
            brne counter

        cpi r28,0                       ; check if the warning message has already
          been sent
```

```
        brne skip1
        rcall temperature
        ldi r25,1                       ; the message comes from INT0
        ldi r28,1                       ; memorize that the warning message has been ↪
          sent
        rcall display
        skip1:                          ; no message is displayed if r28 was already ↪
          set to 1
        reti
uart_rcx:
        rcall UART0_getc

        cpi a0,0b11110101               ; compare register to binary value associated ↪
          to character "u"
        brne not_update
        rcall temperature
        rcall display
        not_update:

        cpi a0,0b11100100               ; compare register to binary value associated ↪
          to character "d"
        brne not_d
        TIMERLIMIT 0,0,0
        ldi r29,0
        not_d:

        cpi a0,0b11110100               ; compare register to binary value associated ↪
          to character "t"
        brne not_t
        TIMERLIMIT 0,0,12
        ldi r29,1
        not_t:

        cpi a0,0b11100110               ; compare register to binary value associated ↪
          to character "f"
        brne not_f
        TIMERLIMIT 0,55,23
        ldi r29,2
        not_f:

        cpi a0,0b11110011               ; Compare register to binary value associated ↪
          to character "s"
        brne not_s
        TIMERLIMIT 55,59,23
        ldi r29,3
        not_s:

        ldi a0,0
        reti

ovf0:
```

```
    inc r22                         ; increment seconds
    cpi r22,60                      ; check if the counter reached one minute
    brne back
    sbi PORTC,0
    ldi r22,0                       ; reset second
    inc r23                         ; increment minute
    cpi r23,60                      ; check if the counter reached one hour
    brne back
    ldi r23,0                       ; reset minutes
    inc r24                         ; increment hours
    cpi r24,24                      ; check if the counter reached one day
    brne back
    ldi r24,0                       ; reset hours
    rcall temperature
    ldi r25,0                       ; message comes from TIMER0
    rcall display

    cpi r29,0                       ; set counter for 24 hours between each report
    brne not_day
    TIMERLIMIT 0,0,0
    not_day:

    cpi r29,1                       ; set counter for 12 hours between each report
    brne not_hours
    TIMERLIMIT 0,0,12
    not_hours:

    cpi r29,2                       ; set counter for 5 minutes between each      ↵
      report
    brne not_minutes
    TIMERLIMIT 0,55,23
    not_minutes:

    cpi r29,3                       ; set counter for seconds between each report
    brne not_seconds
    TIMERLIMIT 55,59,23
    sbi PORTC,1
    not_seconds:
    sbi PORTC,3

    back:

    reti


;====RESET====

reset:
    LDSP    RAMEND                  ; load stack pointer SP
    OUTI DDRC,$ff                   ; set LED
    OUTI DDRD,$00                   ; configure PORTD (buttons) as inputs
    sbi DDRA,SPEAKER                ; make pin SPEAKER an output
```

```
    OUTI EIMSK,0b00000011               ; enable INT0 and INT1
    rcall   wire1_init                  ; initialize 1-wire(R) interface
    rcall   UART0_init                  ; initialize UART
    OUTI TIMSK,1<<TOIE0
    OUTI ASSR,(1<<AS0)
    OUTI TCCR0,5
    ldi r22,0                           ; reset r22
    ldi r25,0                           ; reset r25
    ldi r28,0                           ; reset r28
    ldi r29,0                           ; reset r29

    sei                                 ; set global interrupt

    rjmp main


;====MAIN====

main:

    in r27,MCUCR
    sbr r27,(1<<SE)                     ; enable sleep mode
    out MCUCR, r27
    OUTI TCNT0,1
    sleep

    in r27,MCUCR
    andi r27,~(1<<SE)                   ; disable sleep mode
    out MCUCR, r27

    rjmp    main
```

```
/*
 * personal_library.asm
 *
 *  Authors: Diane Marquette et Luc Reveyron
 */
;====SUB-ROUTINES DEFINITIONS====

;====display=================================================
;purpose: This sub-routine displays a message on the UART terminal and includes the
  current values stored in some registers (temperature,pump's status)
;in:        a0,a1,a2 (temperature)
;           r28 (pump's status: 1 OFF, 0 ON)
;           r21  (message's origin: 1 INT0, 0 TIMER0)
;out:            (a message is displayed on the UART terminal)
;mod:       none
;============================================================
display:
        sbrs r25,0              ; Skip if bit 0 of r25 is set
        rjmp jump1
        PRINTF UART0
.db LF,CR,  "/!\ WARNING /!\",0
        jump1:
        PRINTF UART0
.db LF,CR,  "Report:",0
        PRINTF UART0
.db LF,CR,  "temperature=",FFRAC2+FSIGN,a,4,$42,"C ",CR,0
        PRINTF UART0
.db LF,CR,  "pump's status:",0

        cpi r28,1
        breq skip2              ; Branch to skip2 if r28 = 1
        PRINTF UART0
.db "ON",CR,LF,0
        rjmp jump2
        skip2:
        PRINTF UART0
.db "OFF",CR,LF,0
        jump2:
        ret

;====temperature=============================================
;purpose: This sub-routine asks for the current temperature measured by the probe
;in:        none
;out:       a0,a1,a2
;mod:       a0,a1,a2,c0
;============================================================
temperature:

            rcall   wire1_reset         ; send a reset pulse
            CA  wire1_write, skipROM    ; skip ROM identification
            CA  wire1_write, convertT   ; initiate temp conversion
            WAIT_MS 750                 ; wait 750 msec
```

```
            rcall   wire1_reset         ; send a reset pulse
            CA  wire1_write, skipROM
            CA  wire1_write, readScratchpad
            rcall   wire1_read          ; read temperature LSB
            mov c0,a0
            rcall   wire1_read          ; read temperature MSB
            mov a1,a0
            mov a0,c0

            ret

;====PERSONAL MACRO====

.macro  TIMERLIMIT              ; load the the values we want into r22 (seconds), r23 ↵
  (minutes), r24 (hours)
        ldi r22, @0
        ldi r23, @1
        ldi r24, @2
        .endmacro
.list
```

(*Il s'agit du cahier des charges tel qu'envoyé le 30/04/2018. Certains registres et ports ont été modifiés depuis.*)

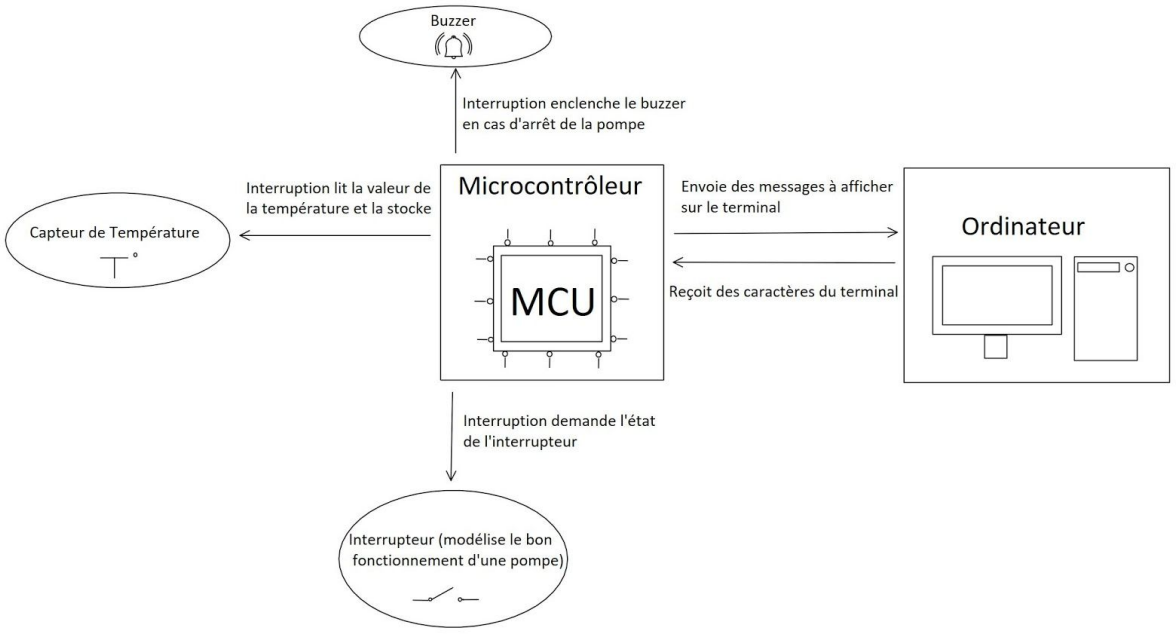
MICRO-PROJET: MONITEUR D'UN DISPOSITIF DE TRAITEMENT DES EAUX USÉES

Présentation générale

Notre projet, doté d'une interface utilisateur (terminal UART), permet de monitorer un dispositif de traitement des eaux usées comportant une pompe. Notre programme:

- lit la température et l'état de la pompe (on = fonctionne, off = panne);
- envoie un message formaté (incluant les dernières données récupérées par les capteurs) toutes les x heures ou minutes;
- envoie un message et génère une alarme sonore (buzzer) quand la pompe tombe en panne;
- permet à l'utilisateur de demander l'envoi immédiat d'un message supplémentaire (taper *u* pour *update*) et de choisir l'intervalle de temps t entre deux messages (parmi les trois durées proposées = 24h (*d* pour *day*), 12h (*t* pour *twelve*) ou 5 min (*f* pour *five*)).

Trois périphériques sont utilisés: le terminal UART, le buzzer et le capteur de température.



Timer

Pour compter le temps écoulé, nous utilisons le timer 0 en choisissant le quartz horloger à 32768 Hz avec un prescaler de 128 (CS0 = 5) pour avoir un overflow toutes les secondes. Nous incrémentons alors les registres nécessaires correspondants aux secondes (r20), aux minutes (r21) et aux heures (r22).

(*Il s'agit du cahier des charges tel qu'envoyé le 30/04/2018. Certains registres et ports ont été modifiés depuis.*)

Interruptions

Trois interruptions sont incluses dans notre programme. La première correspond à l'overflow du timer 0 (INT1). La deuxième (INT1) est enclenchée quand la pompe tombe en panne. Le buzzer se met à sonner et un message est envoyé à l'utilisateur. Une dernière interruption (INT0) permet un reset du dispositif (timer et registres r20, r21, r22 mis à 0 et messages envoyés par défaut toutes les 24h) en appuyant sur le bouton B0.

Sous-routines

ROUTINE	IN	OUT	DESCRIPTION
buzzer		SPEAKER	sous-routine qui est appelée quand la pompe tombe en panne (i.e. l'interrupteur est sur off) et qui génère une alarme sonore pour alerter l'utilisateur
message	- prioritaire / normal (paramètre à 1 ou à 0) - température - état de la pompe (on/off)	chaînes de caractères formatées	un message est affiché sur le terminal UART composé de chaînes de caractères formatées incluant des données collectées par les différents périphériques
read_temperature	valeur du thermomètre indiquée par le capteur	paramètre stockant la valeur de la température en mémoire (r18)	sous-routine qui lit la valeur donnée par le capteur de température et qui la stocke
read_pump	valeur de l'interrupteur (représentant l'état de la pompe)	paramètre stockant l'état de la pompe en mémoire (r19)	sous-routine qui lit l'état de l'interrupteur (on/off) et qui la stocke

Ports

PORT A	PORT B	PORT C	PORT E
buzzer	bouton B0 pour le reset bouton B7 pour la pompe	capteur de température	PE0 -> Receive Data (Rdx) PE1 -> Transmit Data (Tdx)

Luc Reveyron & Diane Marquette