

Correction d'épreuves de la 48^e Compétition Nationale des Métiers

MÉTIER N°16 ELECTRONIQUE

MODULE C1

PROGRAMMATION EMBARQUEE – BAS NIVEAU

CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Soumis par :

Louis LEFEBVRE, Expert WorldSkills France

Dominique CHATEAU, Expert Adjoint WorldSkills France

Référence de la correction : WSFR48CNAT-16-CORRECTION-C1

Révision de la correction : 01

Date de diffusion : POST COMPETITION

OBJECTIFS DU MODULE

L'objectif de ce module est de vérifier l'aptitude des compétiteurs à comprendre le fonctionnement bas niveau d'un microcontrôleur et à programmer celui-ci.

CORRECTION

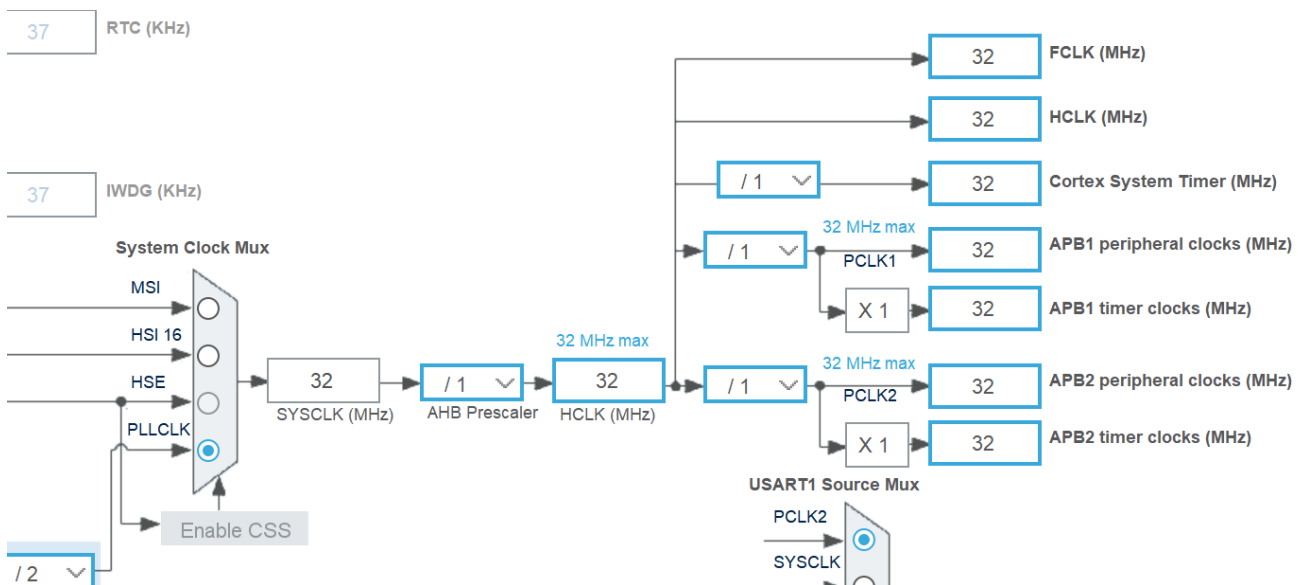
TÂCHE 1 – CONFIGURATION DU TIMER, BOUTONS UTILISATEUR ET AFFICHAGE ECRAN

Configuration du *timer* TIM22

Pour que la LED RGB *LED1* fonctionne, la fréquence de base du *timer* doit être fixée à 24 MHz ou 32 MHz. Le fonctionnement de la LED n'étant cependant pas demandé par le sujet, le choix de la fréquence de base du *timer* TIM22 est laissée libre à l'utilisateur.

Afin de faciliter le travail du compétiteur, l'horloge est déjà partiellement préconfigurée :

- La source de l'horloge est positionnée sur PLLCLK
- La fréquence de base de l'horloge est fixée à 32 MHz



CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Connaissant l'expression de la fréquence de l'interruption d'un *timer* STM32, il est alors attendu que le compétiteur configure le *timer* TIM22 de la façon suivante :

- Clock Source : Internal Clock
- PSC = 7
- ARR = 39999
- TIM22 global interrupt : enabled

$$F_{TIM22}[Hz] = \frac{F_{timer}[Hz]}{(PSC + 1)(ARR + 1)} = \frac{32.10^6}{8 \times 40000} = 100 \text{ Hz}$$

TIM22 Mode and Configuration

Mode

Trigger Source: Disable

Clock Source: Internal Clock

Channel1: Disable

Channel2: Disable

Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bit...): 7

Counter Mode: Up

Counter Period (AutoR...): 39999

Internal Clock Division ...: No Division

auto-reload preload: Disable

Trigger Output (TRGO) Para...

TIM22 Mode and Configuration

Mode

Trigger Source: Disable

Clock Source: Internal Clock

Channel1: Disable

Channel2: Disable

Configuration

Reset Configuration

Parameter Settings | User Constants | NVIC Settings

NVIC Interrupt Table	Enabled	Preemption Priority
TIM22 global interrupt	<input checked="" type="checkbox"/>	0

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

SCR.c

```

/*****
/* @function __SCR_Page_Timers
/*
/* @brief Generates user screen content for "Timers" page.
/* @param [in] Fclk Global source clock frequency
/* @param [in] tim22_timer TIM22 timer handler
/* @param [out] pageContent User screen content
/* @req SYS_REQ-0406-002 : Ecran timers (STATE_TIM)
*****/
static void __SCR_Page_Timers
(
    const double Fclk,
    const TIM_HandleTypeDef * const tim22_timer,
    tScrUsrBuffer pageContent
)
{
    double tim22f;
    tim22f = Fclk / ((tim22_timer->Init.Prescaler + 1) * (tim22_timer->Init.Period + 1));

    snprintf(pageContent[E_SCR_USR_LINE_1], sizeof(tScrUsrBufferLine), "Timer
frequency");
    snprintf(pageContent[E_SCR_USR_LINE_3], sizeof(tScrUsrBufferLine), "TIM22:
%8.2f Hz", tim22f);
}

```

Afin d'activer les interruptions, l'instruction suivante doit être activée dans la fonction **main** :

main.c

```

HAL_TIM_Base_Start_IT(&htim22);

```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Boutons utilisateur

Les boutons `BTN_MODE` et `BTN_DISP` sont respectivement connectés aux *pins* PA2 et PA3 du STM32. Ces *pins* doivent être configurés de la façon suivante :

- *GPIO mode : input mode*
- *No pull-up and no pull-down*

GPIO

ADC

I2C

USART

Search Signals

Search (Ctrl+F)

☐ Show only Modified Pins

Pin Name	Signal on Pin	GPIO output...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PA2	n/a	n/a	Input mode	No pull-up a...	n/a	n/a	BTN_MODE	<input checked="" type="checkbox"/>
PA3	n/a	n/a	Input mode	No pull-up ...	n/a	n/a	BTN_DISP	<input checked="" type="checkbox"/>
PA5	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	LED_RGB	<input checked="" type="checkbox"/>
PA15	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	<input checked="" type="checkbox"/>
PB3	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	<input checked="" type="checkbox"/>
PB4	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	<input checked="" type="checkbox"/>
PB5	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	<input checked="" type="checkbox"/>
PB9	n/a	Low	Output Pus...	No pull-up ...	Low	Disable	RF_CARRI...	<input checked="" type="checkbox"/>

PA2 Configuration :

GPIO mode

Input mode

GPIO Pull-up/Pull-down

No pull-up and no pull-down

User Label

BTN_MODE

Deux solutions s'offrent au compétiteur pour implémenter la détection des boutons telle que demandée :

- **Grâce à la bibliothèque *PushButton* fournie dans le projet** : un compétiteur attentif à son environnement de travail remarquera une bibliothèque « *PushButton* » mise à disposition dans le dossier « *Drivers* ». Cette bibliothèque permet d'instancier immédiatement les boutons de la façon suivante :

main.c

```

tPushButton BtnMode_handler;
tPushButton BtnDisp_handler;

void Init_PushButtons()
{
    PushButton_init(&BtnMode_handler, BTN_MODE_GPIO_Port, BTN_MODE_Pin,
PBM_RISING_EDGE);
    PushButton_init(&BtnDisp_handler, BTN_DISP_GPIO_Port, BTN_DISP_Pin,
PBM_RISING_EDGE);
}

```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

```
void PushButtons_ReadCurrentStates
(
    bool * const V_BTN_MODE,
    bool * const V_BTN_DISP
)
{
    /* ---- Temporary disabling TIM22 interrupts to copy button states ---- */
    HAL_NVIC_DisableIRQ(TIM22_IRQn);

    /* ---- Setting button status output ---- */
    *V_BTN_MODE = (BtnMode_handler.lstate == PBLIS_ACTIVE);
    *V_BTN_DISP = (BtnDisp_handler.lstate == PBLIS_ACTIVE);

    /* ---- Acknowledgement of rising edge detection ---- */
    BtnMode_handler.edge_ack = true;
    BtnDisp_handler.edge_ack = true;

    /* ---- Re-enabling TIM22 interrupts ---- */
    HAL_NVIC_EnableIRQ(TIM22_IRQn);
}
```

stm32l0xx_it.c

```
void TIM22_IRQHandler(void)
{
    /* USER CODE BEGIN TIM22_IRQn 0 */

    /* ===== */
    /* Button state reading frequency */
    /* ===== */
    /* Button states are read by TIM22. TIM22 interrupt frequency is set by */
    /* MX_TIM22_Init and button states are read by TIM22_IRQHandler function. */
    /* TIM22 config is set as follows: */
    /* - Fclk = 32 MHz */
    /* - PSC = 7 */
    /* - ARR = 39999 */
    /* Finterrupt = Fclk / ((PSC+1)(ARR+1)) = 32.10^6/(8*40000) = 100 Hz */
    /* => T = 10 ms */
    /* ----- */

    PushButton_read(&BtnMode_handler);
    PushButton_read(&BtnDisp_handler);

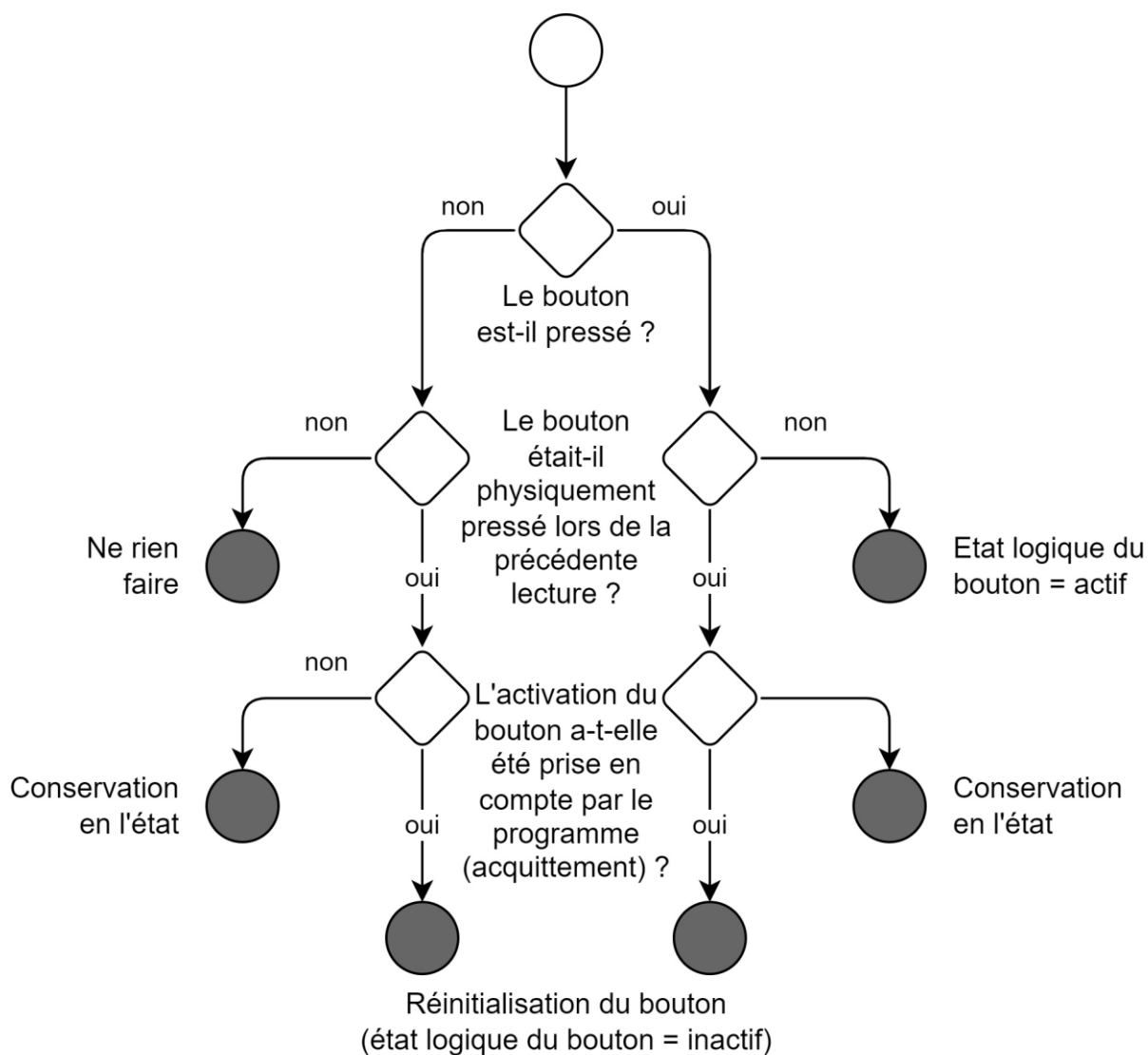
    /* USER CODE END TIM22_IRQn 0 */
    HAL_TIM_IRQHandler(&htim22);
    /* USER CODE BEGIN TIM22_IRQn 1 */

    /* USER CODE END TIM22_IRQn 1 */
}
```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

- Implémentation complète** : si le compétiteur ne remarque pas la bibliothèque ou n'arrive pas à l'utiliser, la deuxième solution est d'implémenter le fonctionnement des boutons à la main. Pour répondre au cahier des charges de boutons pris en compte seulement sur front montant de leur activation, l'implémentation doit suivre la logique suivante :

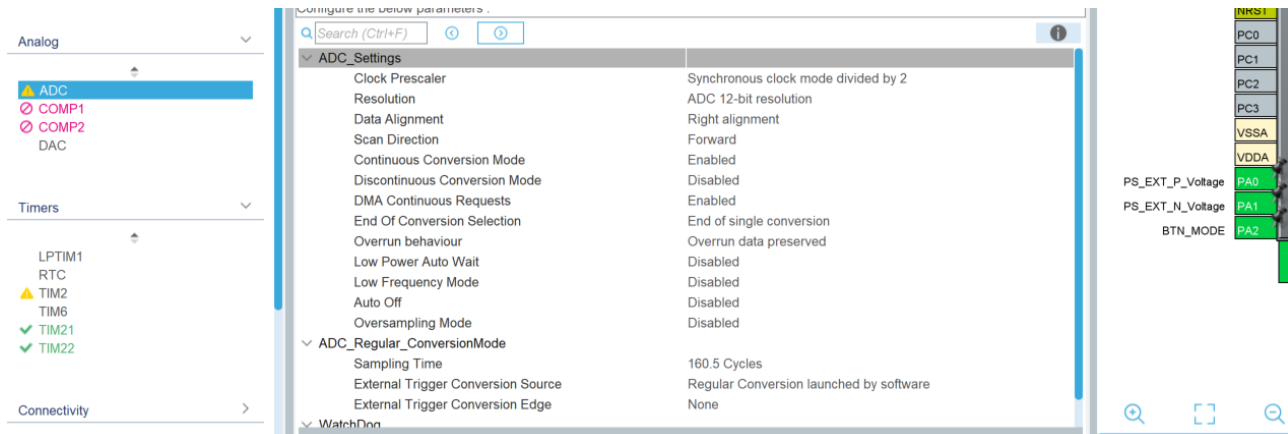


CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

TÂCHE 2 – TENSIONS D'ALIMENTATION

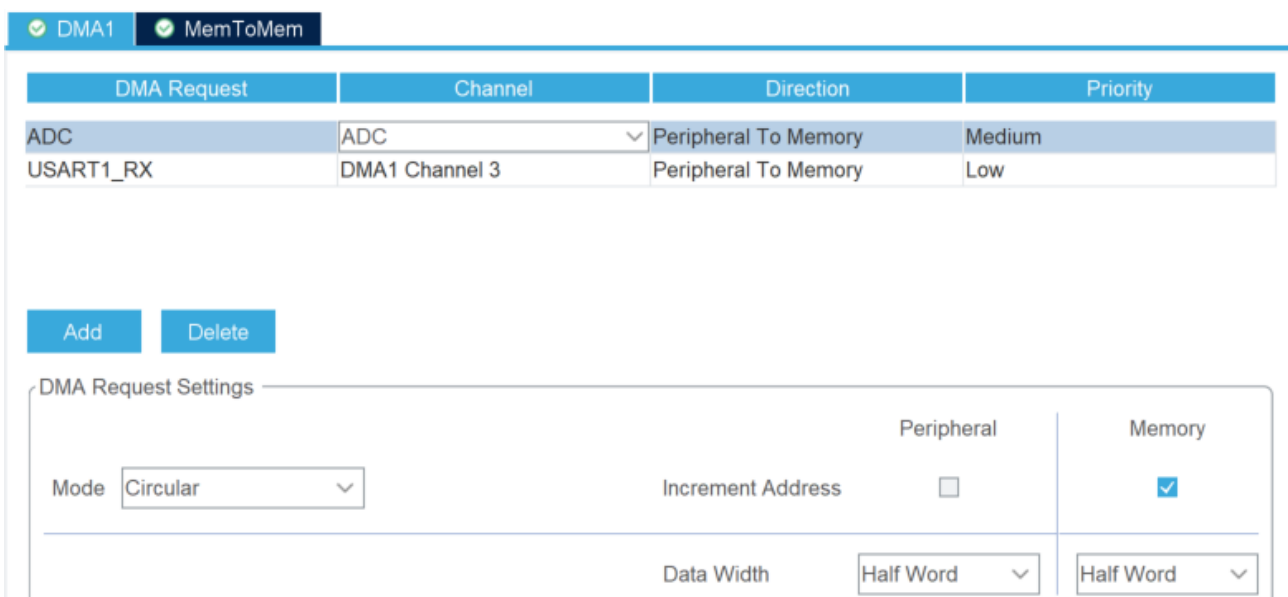
La lecture des tensions d'alimentation nécessite la configuration du convertisseur analogique → numérique (ADC) sur les pins PA0 et PA1.



Deux méthodes sont possibles pour lire les valeurs de l'ADC :

- **Par polling** : ne peut lire qu'un canal de l'ADC à la fois, et nécessite une reconfiguration de l'ADC chaque fois que l'on souhaite changer de canal. En plus de ces opérations, la méthode par *polling* est longue car n'exécute la lecture de la valeur que sur demande.
- **Par direct memory access (DMA)** : les canaux ADC sont lus en boucle par le matériel (pas par le logiciel), et les valeurs lues sont directement recopiées en mémoire volatile et disponibles à tout instant. Cette méthode est plus complexe à mettre en place, mais ne nécessite pas de reconfigurer l'ADC pour changer de canal, les valeurs sont toujours disponibles immédiatement et la fonctionnalité ne consomme pas de charge CPU.

Pour cette correction, la méthode choisie est le DMA.



CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Contexte de la lecture des alimentations

PWR.c

```
#define C_PWR_ADC_MAX_VALUE ((double)4095.0) /* 12-bit ADC */
#define C_PWR_PS_EXT_P_MAX_VOLTAGE ((double) 9.0) /* PS_EXT_P = [ 0 ; 9] V */
#define C_PWR_PS_EXT_N_MAX_VOLTAGE ((double)-9.0) /* PS_EXT_N = [-9 ; 0] V */

typedef enum
{
    E_POWERSOURCE_P = 0,
    E_POWERSOURCE_N,
    E_NB_POWERSOURCE
} tPWR_PowerSource;

static unsigned int __PWR_POWERSOURCE_ADC_CHANNELS[E_NB_POWERSOURCE];

static ADC_HandleTypeDef * __PWR_ADC_HANDLER;

/* Warning: DMA is configured in half-word values (16 bit values) */
static volatile uint16_t __PWR_ADC_DMA_VALUES[E_NB_POWERSOURCE];
```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Initialisation du DMA

PWR.c

```

/*****
/* @function PWR_Init
/*
/* @brief Initializes power supply monitoring context.
/* @param [in|out] adcHandler Analog to digital converter handler
/* @retval Result value of HAL_ADC_Start_DMA function.
/* @req None
*****/
HAL_StatusTypeDef PWR_Init(ADC_HandleTypeDef * const adcHandler)
{
    tPWR_PowerSource ipwrs;
    HAL_StatusTypeDef retval;

    __PWR_ADC_HANDLER = adcHandler;

    __PWR_POWERSOURCE_ADC_CHANNELS[E_POWERSOURCE_P] = ADC_CHANNEL_0;
    __PWR_POWERSOURCE_ADC_CHANNELS[E_POWERSOURCE_N] = ADC_CHANNEL_1;

    /* ----- */
    /* Power supply monitoring is done by reading power supply voltages.
    /* To do so, the ADC (analog to digital converter) is configured to read
    /* both positive (PS_P_Voltage, on channel ADC_CHANNEL_0) and negative
    /* (PS_N_Voltage, on channel ADC_CHANNEL_1) board power sources.
    /*
    /*
    /* There are two methods to read the ADC values :
    /* - Polling method, which can only read one channel at a time and
    /* requires to reconfigure ADC each time to change the channel read.
    /* - Continuous reading with DMA copy, which continuously reads the ADC
    /* values and store them in an array.
    /*
    /*
    /* The DMA method is more complex than the polling method, but have many
    /* benefits:
    /* - There is no need to reconfigure the ADC each time, which reduces CPU
    /* usage as less instructions are performed.
    /* - The values are always available and there is no need to wait for the
    /* ADC to read them (the reading is done in background, while polling
    /* method starts to read the values only when asked to).
    /* - The cyclic reading is a hardware feature, and DMA only copies values
    /* to RAM when they are ready, which means no CPU usage at all!
    /* ----- */

    for(ipwrs = 0 ; ipwrs < E_NB_POWERSOURCE ; ++ipwrs)
    { PWR_ADC_DMA_VALUES[ipwrs] = 0; }

    retval = HAL_ADC_Start_DMA(__PWR_ADC_HANDLER,
    (uint32_t*)__PWR_ADC_DMA_VALUES, E_NB_POWERSOURCE);
    return retval;
}

```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Lecture des valeurs de l'ADC

PWR.c

```

/*****
/* @function PWR_ReadPowerSourceVoltage */
/* */
/* @brief Reads the voltage of a board power supply. */
/* @param [in] powerSource Power supply source to read */
/* @retval Power supply voltage if the power source is known; NaN otherwise. */
/* @req SYS_REQ-0106-001 : Numérisation de la tension d'alimentation positive */
/* @req SYS_REQ-0107-001 : Numérisation de la tension d'alimentation négative */
*****/
double PWR_ReadPowerSourceVoltage(const tPWR_PowerSource powerSource)
{
    double rawAdcValue;
    double powerSourceVoltage;

    /* ----- */
    /* Note: as the DMA directly copies the ADC values into RAM in a */
    /* user-defined variable (here __PWR_ADC_DMA_VALUES), there is a risk */
    /* that the program reads the values at the same time they are written, */
    /* which may lead to inconsistent data read. To prevent this behavior, */
    /* the HAL_ADC_ConvCpltCallback callback function can be defined to store */
    /* the values in non volatile memory upon DMA copy. However, there is no */
    /* need for precise reading in this application and an occasional reading */
    /* error is acceptable, so this method is not implemented. */
    /* ----- */
    /* Note: the ADC sampling duration is not guaranteed, so the data stored */
    /* in __PWR_ADC_RAW_VALUES may not always be up to date. */
    /* To ensure periodic ADC reading, a method might be to poll ADC values */
    /* during a periodic interruption instead of asking the hardware to do */
    /* the job on its own. However, there is no need for precise reading in */
    /* this application, so this method is not implemented. */
    /* ----- */
    switch(powerSource)
    {
        case E_POWERSOURCE_P:
            /* PS_EXT_P = [0;9] V converted to PS_EXT_P_Voltage = [0;3.3] V */
            /* in order to be numerized by the ADC. As the [0;9]->[0;3.3] */
            /* value mapping is linear, and as 3.3V is the maximum value the */
            /* ADC can read, the [0;3.3] mapping is transparent for the */
            /* ADC raw value to [0;9] voltage conversion. */
            rawAdcValue = (double)__PWR_ADC_DMA_VALUES[E_POWERSOURCE_P];
            powerSourceVoltage = (rawAdcValue / C_PWR_ADC_MAX_VALUE) *
C_PWR_PS_EXT_P_MAX_VOLTAGE;
            break;
    }
}

```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

```

        case E_POWERSOURCE_N:
            /* PS_EXT_N = [-9;0] V converted to PS_EXT_N_Voltage = [0;3.3] V */
            /* in order to be numerized by the ADC. As the [-9;0]->[0;3.3] */
            /* value mapping is symetric-linear, and as 3.3V is the maximum */
            /* value the ADC can read, the [0;3.3] mapping is transparent for */
            /* the ADC raw value to [0;9] voltage conversion. */
            rawAdcValue = (double)__PWR_ADC_DMA_VALUES[E_POWERSOURCE_N];
            powerSourceVoltage = (rawAdcValue / C_PWR_ADC_MAX_VALUE) *
C_PWR_PS_EXT_N_MAX_VOLTAGE;
            break;

        default:
            powerSourceVoltage = NAN;
            break;
    }

    return powerSourceVoltage;
}

```

Affichage des valeurs

SCR.c

```

/*****
/* @function __SCR_Page_Power
/*
/* @brief Generates user screen content for "Power" page.
/* @param [in] ps_p_voltage Positive power source voltage
/* @param [in] ps_n_voltage Negative power source voltage
/* @param [out] pageContent User screen content
/* @req SYS_REQ-0404-001 : Ecran alimentation (STATE_PWR)
*****/
static void __SCR_Page_Power
(
    const double ps_p_voltage,
    const double ps_n_voltage,
    tScrUsrBuffer pageContent
)
{
    snprintf(pageContent[E_SCR_USR_LINE_1], sizeof(tScrUsrBufferLine), "Power
voltages ");
    snprintf(pageContent[E_SCR_USR_LINE_2], sizeof(tScrUsrBufferLine), "PS_EXT_P:
%6.3f V", ps_p_voltage);
    snprintf(pageContent[E_SCR_USR_LINE_3], sizeof(tScrUsrBufferLine), "PS_EXT_N:
%6.3f V", ps_n_voltage);
}

```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

TÂCHE 3 – TRANSMISSION DES DONNEES RF

Activation de la génération de la porteuse

L'activation de la porteuse est effectuée en configurant la sortie `RF_CARRIER_ACTIVE` (PB9) en sortie et en la positionnant à l'état haut lorsque la porteuse est active, et à l'état bas lorsqu'elle est inactive.

Pin Name	Signal on Pin	GPIO output...	GPIO mode	GPIO Pull-u...	Maximum o...	Fast Mode	User Label	Modified
PA2	n/a	n/a	Input mode	No pull-up ...	n/a	n/a	BTN_MODE	✓
PA3	n/a	n/a	Input mode	No pull-up ...	n/a	n/a	BTN_DISP	✓
PA5	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	LED_RGB	✓
PA15	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓
PB3	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓
PB4	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓
PB5	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓
PB9	n/a	Low	Output Pus...	No pull-up ...	Low	Disable	RF_CARRI...	✓
PC10	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓
PC11	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓
PC12	n/a	Low	Output Pus...	No pull-up ...	Low	n/a	COM_ADD...	✓

PB9 Configuration :

GPIO output level	Low
GPIO mode	Output Push Pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	Low
Fast Mode	Disable

COM_RfCarrier.c

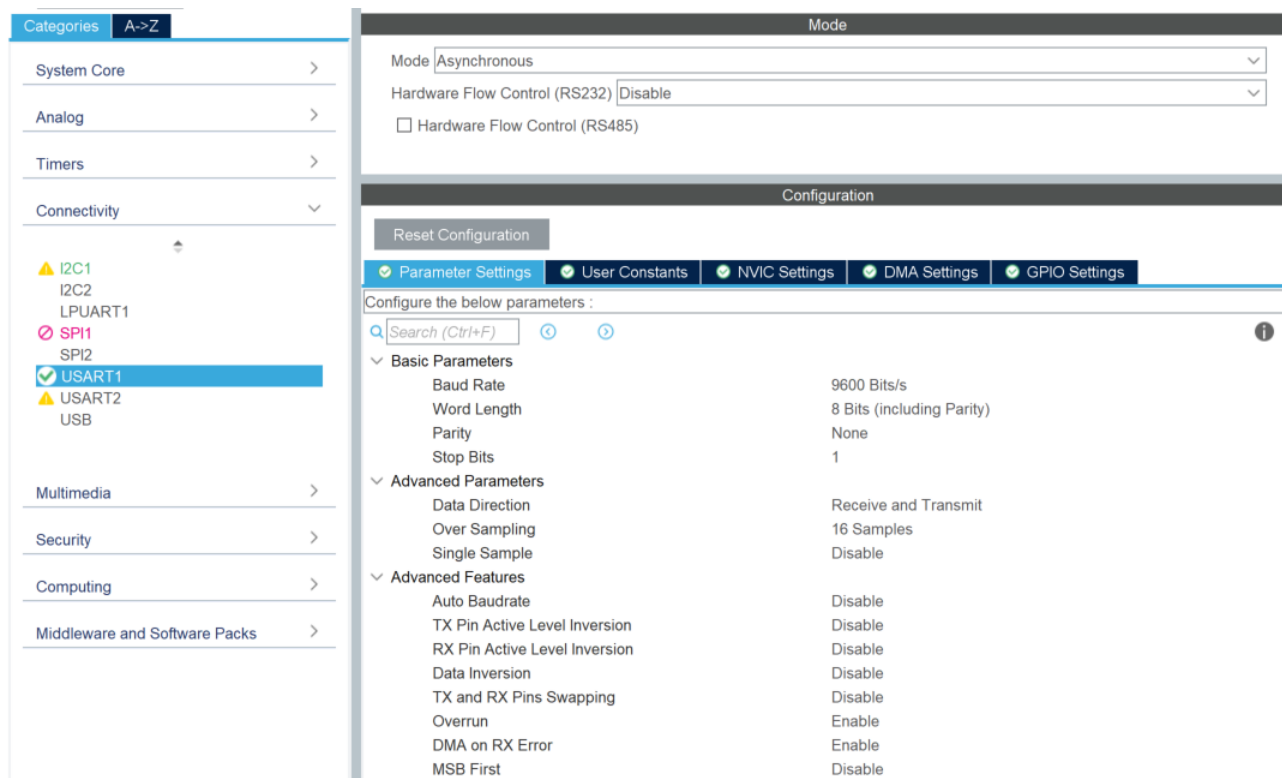
```
void COM_ManagerRfCarrier(const bool rfCarrierActivationStatus)
{
    tCOM_EpromBits bit;

    if(rfCarrierActivationStatus)
    {
        HAL_GPIO_WritePin(RF_CARRIER_ACTIVE_GPIO_Port, RF_CARRIER_ACTIVE_Pin,
GPIO_PIN_SET);
    }
    else
    {
        HAL_GPIO_WritePin(RF_CARRIER_ACTIVE_GPIO_Port, RF_CARRIER_ACTIVE_Pin,
GPIO_PIN_RESET);
    }
}
```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

Configuration de la liaison série permettant l'envoi des données



Sélection du facteur de modulation

Afin de sélectionner le facteur de modulation, il suffit de configurer les pins *COM_ADDR_BIT_xx* en sorties et d'écrire l'état du bit d'adresse correspondant.

Lorsque la porteuse est active, les adresses du facteur de modulation dans l'EPROM sont respectivement 0x72 et 0x73 pour l'état bas et l'état haut. Ces deux adresses sont successives, c'est-à-dire que l'état haut ou bas est choisi par le bit 0, relié à *RS_DATA*. Il suffit alors d'écrire les bits de « 0x72 » (en binaire : 0111 0010) sur les pins restants :

Pin	Décalage de bit	Etat du pin
COM_ADDR_BIT_00	RS_DATA	
COM_ADDR_BIT_01	(0x72 >> 1) & 0x01	1
COM_ADDR_BIT_02	(0x72 >> 2) & 0x01	0
COM_ADDR_BIT_03	(0x72 >> 3) & 0x01	0
COM_ADDR_BIT_04	(0x72 >> 4) & 0x01	1
COM_ADDR_BIT_05	(0x72 >> 5) & 0x01	1
COM_ADDR_BIT_06	(0x72 >> 6) & 0x01	1
COM_ADDR_BIT_07	(0x72 >> 7) & 0x01	0

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.

COM_RfCarrier.c

```

/*****
/* @function COM_ManageRfCarrier
/*
/* @brief Manages the RF transmission carrier.
/* @param [in] rfCarrierActivationStatus RF carrier activation command
/* @pre The RF carrier control context must have been initialized by
/* COM_RfCarrier_Init
/* @req SYS_REQ-0603-001 : Redressage du signal RS_DATA
/* @req SYS_REQ-0604-001 : Génération de la porteuse
*****/
void COM_ManageRfCarrier(const bool rfCarrierActivationStatus)
{
    tCOM_EpromBits bit;

    if(rfCarrierActivationStatus)
    {
        /* ----- */
        /* Note: The modulation factors stored in EPROM are placed one after */
        /* the other. The __C_COM_EPROM_ADDR_MODULATION_FACTOR address stores */
        /* the low state value, and __C_COM_EPROM_ADDR_MODULATION_FACTOR + 1 */
        /* stores the high state value. */
        /* The address COM_ADDR_BIT_00 is therefore never directly written by */
        /* this function, but is set by the RS_DATA signal. */
        /* ----- */

        for(bit = E_COM_EPROM_ADDR_BIT_01 ; bit < E_COM_EPROM_ADDR_NB_BITS ; ++bit)
        {
            HAL_GPIO_WritePin
            (
                C_COM_EPROM_ADDR[bit].gpioPort,
                C_COM_EPROM_ADDR[bit].gpioPin,
                (__C_COM_EPROM_ADDR_MODULATION_FACTOR >> bit) & 0x01
            );
        }
    }
    else
    {
        for(bit = E_COM_EPROM_ADDR_BIT_01 ; bit < E_COM_EPROM_ADDR_NB_BITS ; ++bit)
        {
            HAL_GPIO_WritePin
            (
                C_COM_EPROM_ADDR[bit].gpioPort,
                C_COM_EPROM_ADDR[bit].gpioPin,
                GPIO_PIN_RESET
            );
        }
    }
}

```

CONFIDENTIEL – CORRECTION

Ce document contient la correction d'un sujet de compétition. Il est strictement confidentiel et ne doit pas être divulgué aux compétiteurs avant la fin de la compétition.