5TC option Embed

OS embarqués légers, contrôle du composant radio

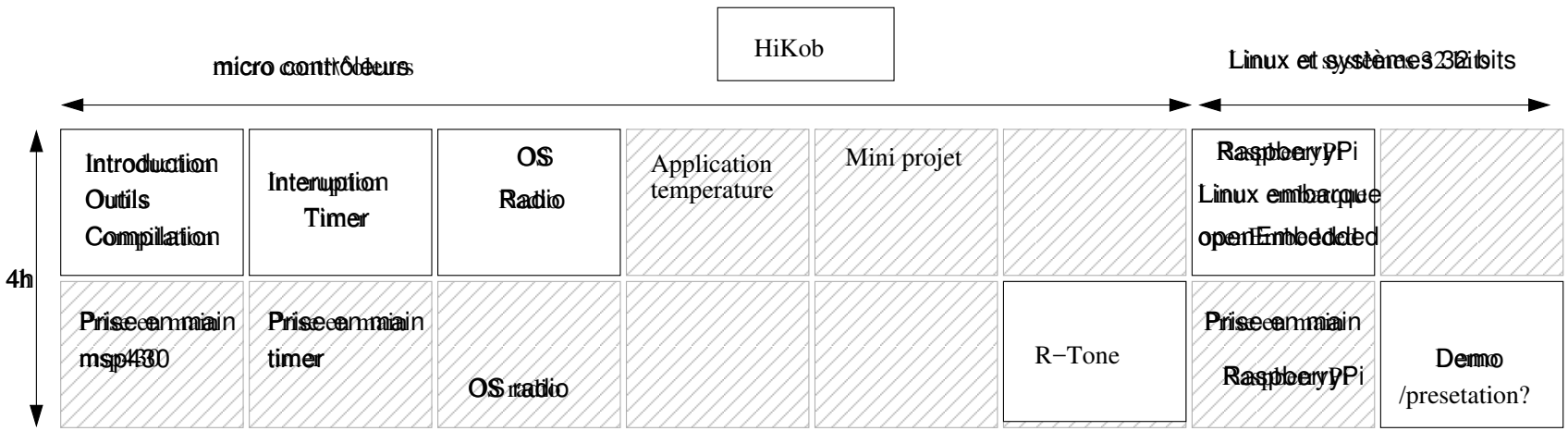
Kevin Marquet, Tanguy Risset, Guillaume Salagnac

prénom.nom@insa-lyon.fr

(Précédents contributeurs : Antoine Fraboulet, Antoine Scherrer, Sylvain Geneves) Labo CITI, INSA de Lyon, Dpt Télécom

18 septembre 2017

Embed planning



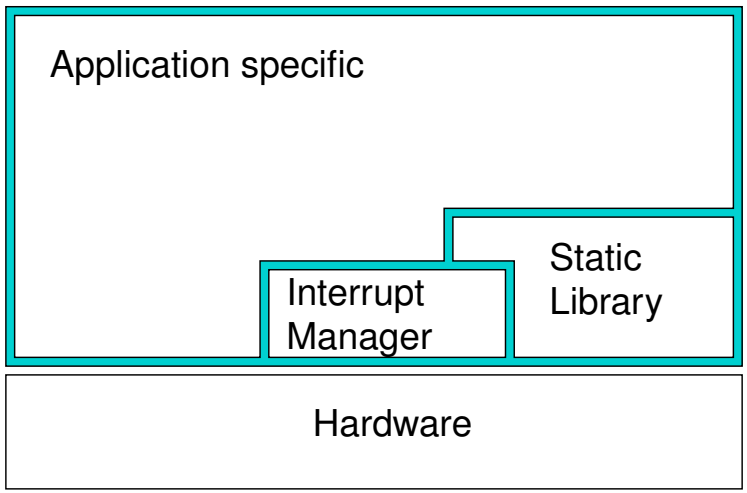
Section 1

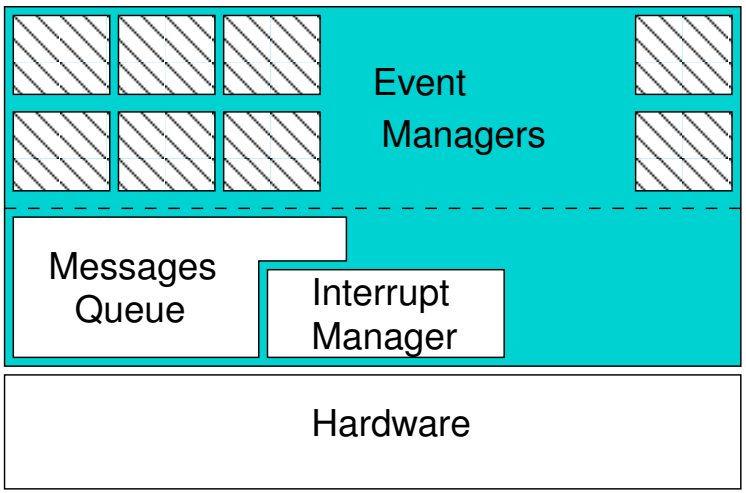
OS embarqués légers

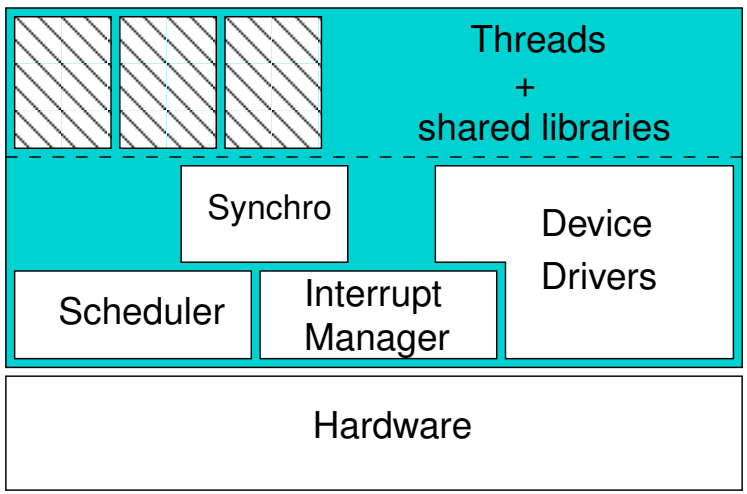
Systèmes d'exploitation légers

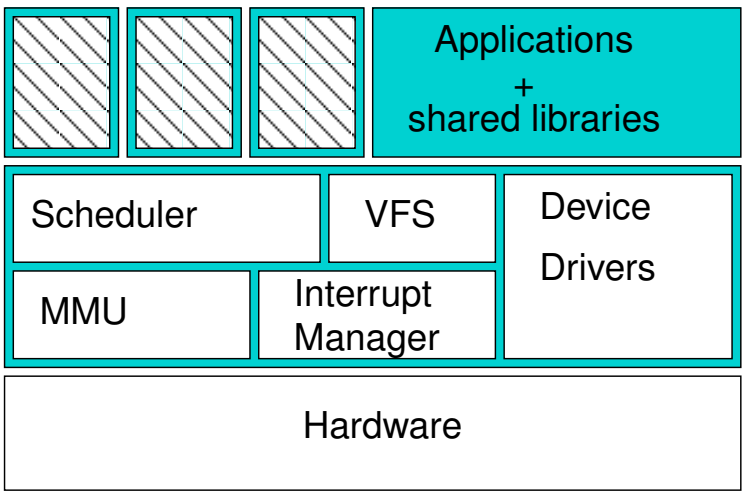
- Les systèmes d'exploitation peuvent aller d'une bibliothèque spécifique pour une application à un système générique type Unix.
- Les applications sans système d'exploitation représentent une part importante des systèmes déployés aujourd'hui.
- Il existe tout de même deux grandes catégories de système
  - modèle "Event driven"
  - modèle "Thread"

Catégories des systèmes









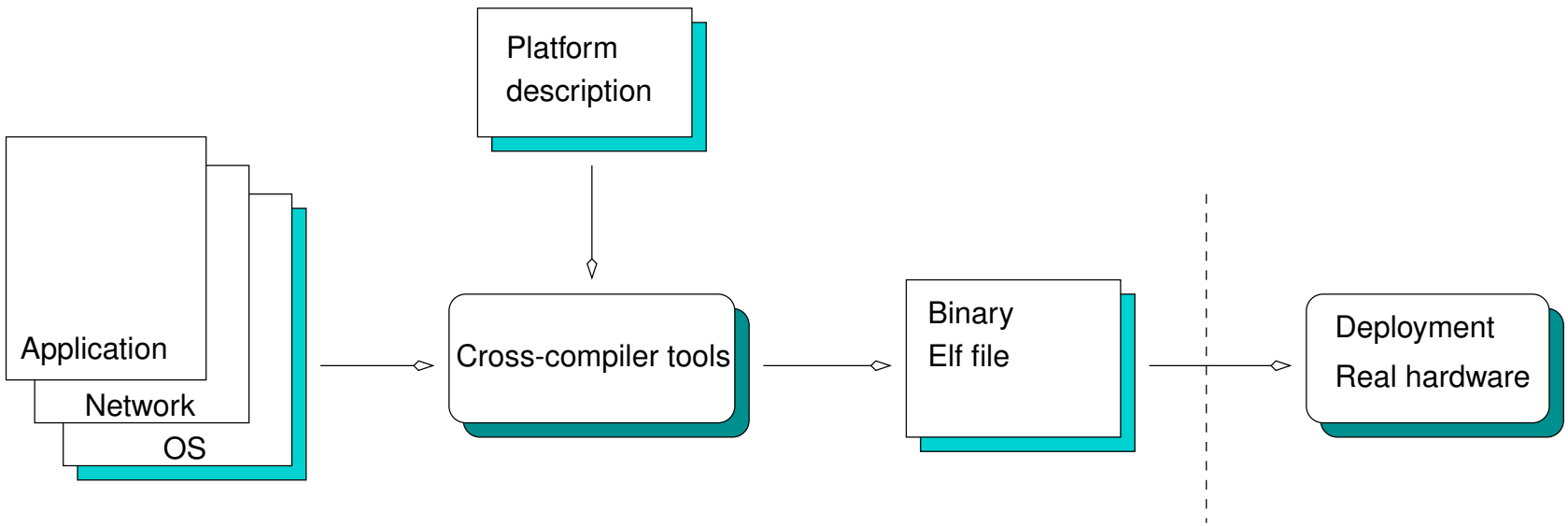
Modèles de programmation et d'exécution

- Evénements :
  - Les événements matériels démarrent des fonctions qui s'exécutent sans interruption (*run to completion*).
  - Les changements de contexte, la gestion de pile, l'ordonnancement et la gestion de priorité sont simplifiés.
  - Exemples : TinyOS 1 & 2

- File de programme / *Thread* :
  - Proche du modèle de programmation classique.
  - Mémoire partagée, piles séparées.
  - Changement de contexte.
  - Exemples : FreeRTOS, Contiki

Environnement logiciel

Les applications sont souvent simples. Les deux modèles sont fait pour être liés statiquement au programme et embarqués dans le système.



Pourquoi utiliser un OS ?

Quels services demander à un OS ?

- Gestion de Tâches/Files = ordonnanceur
- Pilotes de périphériques = interface matériel
  - Gestionnaire d'interruption
  - Gestion du temps et des timers

- Gestion des modes de veille
- Pile réseau intégrée

- Environnement de programmation et outils

Aperçu des systèmes

3 exemples de systèmes utilisés dans les petits objets.

1. TinyOS : modèle à événements
2. FreeRTOS : modèle à thread
3. Contiki : modèle à protothread (Anciennement Co-routine)

Événements : exemple TinyOS

- Gestion des événements
- Fréquences fixes, mode basse conso simple
- Propose des abstractions pour
  - les communications
  - les timers
  - le stockage

- Modèle d'exécution : run to completion
- **Utilisation d'une seule pile d'exécution**

- TinyOS 2.x légère amélioration du système de mise en veille

TinyOS 1.x main loop (1/2)

```
int main(void)
{
  MainM$hardwareInit();
  TOSH_sched_init();
  MainM$StdControl$init();
  MainM$StdControl$start();
  __nesc_enable_interrupt();
  for (; ; ) {
    TOSH_run_task();
  }
}
```

TinyOS 1.x main loop (2/2)

```
bool TOSH_run_task(void)
{
  void (*func)(void );
  __nesc_atomic_t fInterruptFlags = __nesc_atomic_start();
  uint8_t                  old_full = TOSH_sched_full;
  func = TOSH_queue[old_full].tp;
  if (func == NULL) {
      __nesc_atomic_sleep();
      return 0;
  }
  TOSH_queue[old_full].tp = NULL;
  TOSH_sched_full = (old_full + 1) & TOSH_TASK_BITMASK;
  __nesc_atomic_end(fInterruptFlags);
  func();
  return 1;
}
```

Préemption : exemple FreeRTOS 4.x

- Opérations de base :
  - Gestion de tâches
  - **Ordonnancement préemptif**
  - Timers & Synchronisation (mutex)

- Utilisation de priorités
- Ordonnancement préemptif
- Primitives de synchronisation
- **Piles séparées par thread**
- Tâche “idle” de plus faible priorité

- Pas de pilote de périphérique

FreeRTOS 4.x main loop (1/2)

```
portTASK_FUNCTION(task_periodic_send, pvParameters) {
    const portTickType xDelay = 1000 / portTICK_RATE_MS;
    for(;;) {
        send_temperature();
        vTaskDelay(xDelay);
    }
}

int main( void ) {
  prvSetupHardware();
  vParTestInitialise();   // start Idle Task
  xTaskCreate(task_periodic_send, "RADIO", STACK_SIZE,
        & ParameterToPass, TASK_PRIORITY, &task_handle );
  vTaskStartScheduler(); // never returns
  return 0;
}
```

FreeRTOS 4.x main loop (2/2)

```
interrupt (TIMERA0_VECTOR) prvTickISR( void )
{
  portSAVE_CONTEXT();
  vTaskIncrementTick();
  vTaskSwitchContext();
  portRESTORE_CONTEXT();
}
```

Cette interruption est appelée périodiquement par un *timer*.

Coroutines / Protothread : exemple Contiki

Coroutines

- Multi-tâche coopératif.
- L'application reste maître de l'ordonnancement.

Protothread

- Modèle très proche des coroutines
- Modèle mixte, orienté événements
  - "run to completion"
  - **Changement de fil sur opération bloquante**
  - **Pile d'exécution unique**
  - Les "thread" n'ont pas d'état (variables locales)

Contiki 2.x main loop (1/2)

```
int main(void) {
  init();
  while (1) {
        do {
            mythread();   /* Thread activation */
        } while (process_run() > 0);

        /* Idle processing. Disable interrupts */
        if (process_nevents() != 0) {
            /* Re-enable interrupts and handle event */
        } else {
            /* Re-enable interrupts and go to sleep atomically. */
        }
  }
  return 0;
}
```

Contiki 2.x protothread (2/2)

```
#define PT_WAIT_UNTIL(pt, condition)     \
  do {                                   \
    LC_SET((pt)->lc);                    \
    if(!(condition)) {                   \
      return PT_WAITING;                 \
    }                                    \
  } while(0)
static PT_THREAD(thread_periodic_send(struct pt *pt))  {
    PT_BEGIN(pt);
    while(1)  {
        TIMER_RADIO_SEND = 0;
        PT_WAIT_UNTIL(pt, node_id != NODE_ID_UNDEFINED &&
                  timer_reached( TIMER_RADIO_SEND, 1000));
        send_temperature();
    }
    PT_END(pt);
}
```

Contiki 2.x protothread (Extra)

```
/* LC = Local Continuation */

#define PT_BEGIN(pt) { char PT_YIELD_FLAG = 1;
LC_RESUME((pt)->lc)

#define PT_END(pt) LC_END((pt)->lc); PT_YIELD_FLAG = 0; \
                   PT_INIT(pt); return PT_ENDED; }
```

Machine de Duff (1/2)

En 1984 Tom Duff travaillant pour LucasFilm cherche à accélérer le code suivant :

```
send(int *to, int *from, int count)
        {
                do
                        *to = *from++;
                while(--count>0);
        }
```

Machine de Duff 2/2)

```
send(int *to, int *from, int count)
{
    register n=(count+7)/8;
    switch(count%8){
        case 0: do{ *to++ = *from++;
        case 7:  *to++ = *from++;
        case 6:  *to++ = *from++;
        case 5:  *to++ = *from++;
        case 4:  *to++ = *from++;
        case 3:  *to++ = *from++;
        case 2:  *to++ = *from++;
        case 1:  *to++ = *from++;
                   }while(--n>0);
    }
}
```

http://www.lysator.liu.se/c/duffs-device.html

Protothread Adam Dunkel : lc-switch.h

```
#define LC_INIT(s) s = 0;

#define LC_RESUME(s) switch(s) { case 0:

#define LC_SET(s) s = __LINE__; case __LINE__:

#define LC_END(s) }
```

Protothread Adam Dunkel : pt.h

```
#include "lc.h"

typedef unsigned short lc_t;
struct pt {
  lc_t lc;
};
#define PT_INIT(pt)   LC_INIT((pt)->lc)
#define PT_THREAD(name_args) char name_args
#define PT_BEGIN(pt) { char PT_YIELD_FLAG = 1;
LC_RESUME((pt)->lc)

#define PT_WAIT_UNTIL(pt, condition)          \
  do {       \
    LC_SET((pt)->lc);     \
    if(!(condition)) {     \
      return PT_WAITING;    \
    }       \
  } while(0)

#define PT_END(pt) LC_END((pt)->lc); PT_YIELD_FLAG = 0; \
                   PT_INIT(pt); return PT_ENDED; }
```

Section 2

SPI et CC2500

Piloter la radio

- Le MSP430 pilote le chip radio via le protocole SPI. Deux types d'actions possibles :
  - Configuration du chip radio
  - envoi/reception de paquet
- Pour la carte MSP-EXP430F5438
  (source `MSP430F5438-Exp-BOARD-User-guide-slau263g.pdf` p22)
  - la communication se fait en utilisant le module USCI **UCB0** configuré en mode SPI.
  - P3.0 : UCB0STE (RF_STE)
  - P3.1 : UCB0SIMO (RF_MOSI)
  - P3.2 : UCB0SOMI (RF_MISO)
  - P3.3 : UCB0CLK (RF_SPI_CLK)

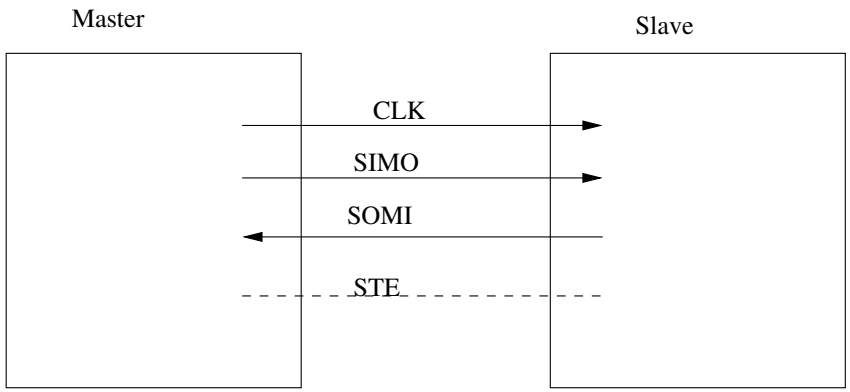
Périphérique USCI

Source : `MSP430F5XX-family-user-guide-slau208k.pdf` chap 35 (p. 923) : Universal Serial Communication Interface – SPI Mode.

- Le périphérique USCI (universal serial communication interface) supporte différent mode de communication série
- Il existe deux sorte de périphérique USCI : USCI_A et USCI_B.
- Un modèle de MSP possède plusieurs modules USCI, on les notes alors, USCI_A0, USCI_A1, USCI_A2, etc..
- Les modules USCI_Ax supportent :
  - mode UART
  - Pulse shaping (communications IrDA)
  - Automatic baud-rate detection for LIN communications
  - mode SPI
- Les modules USCI_Bx supportent :
  - mode I2C
  - mode SPI

SPI

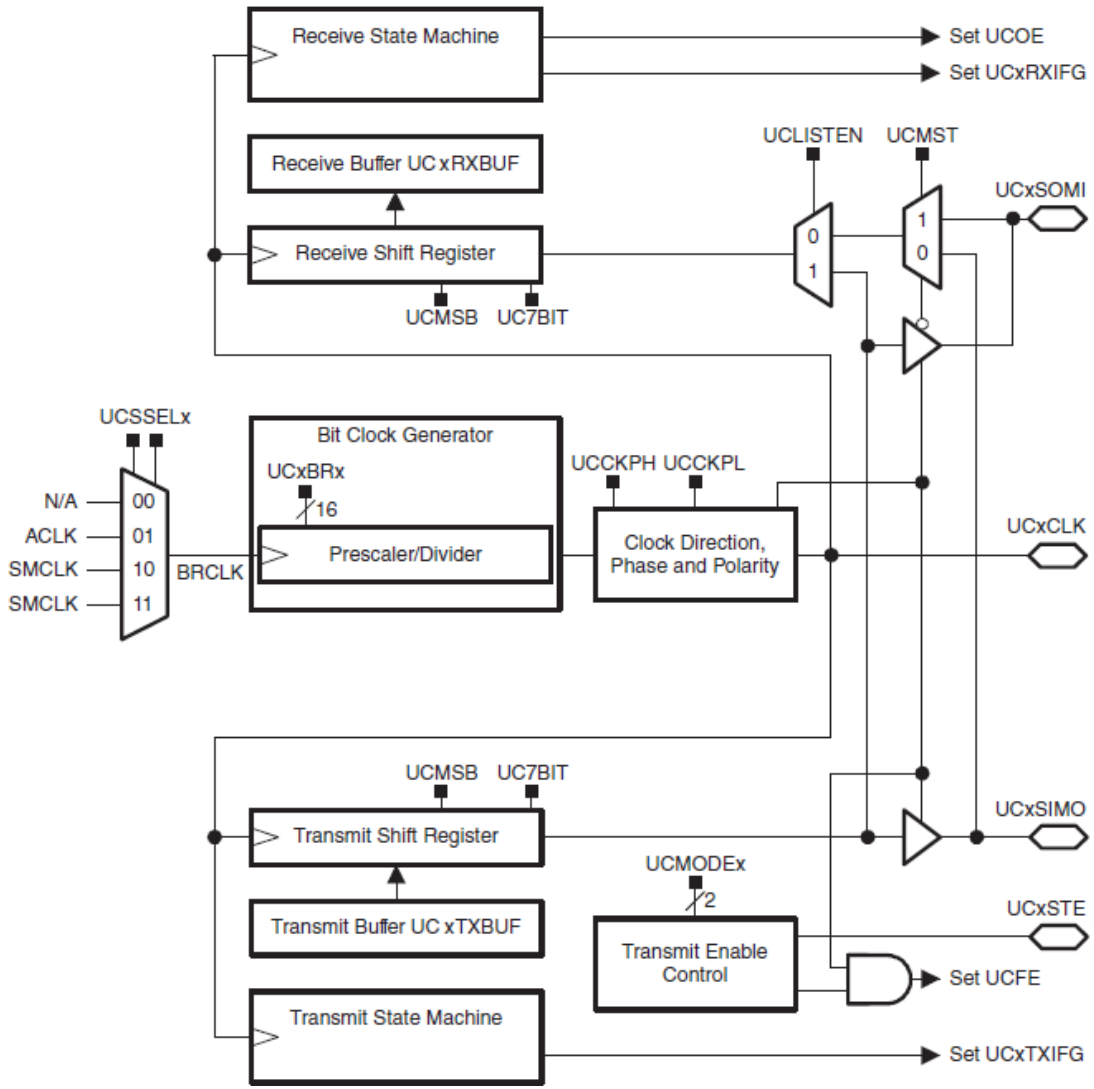
- Le protocole SPI est un protocole synchone qui connecte un device à un ou plusieurs autres grâce à 3 ou 4 fils :
- Sur un module USCI, le protocole SPI est selectionné quand le bit UCSYNC est activé et le mode SPI (3 ou 4 fils, selectionné par UCMODEx)
- Ces quatres fils sont :
  - UCxSIMO (Slave In, Master Out)
  - UCxSOMI (Slave Out Master In)
  - UCxCLK
  - UCxSTE (Slave Trasmit enable permet plusieurs Master sur un seul bus SPI)



SPI

- SPI mode features include :
  - 7-bit or 8-bit data length
  - LSB-first or MSB-first data transmit and receive
  - 3-pin and 4-pin SPI operation
  - Master or slave modes
  - Continuous transmit and receive operation
  - Selectable clock polarity and phase control
  - Programmable clock frequency in master mode
  - Independent interrupt capability for receive and transmit
  - Slave operation in LPM4

USCI Block diagram : mode SPI



Séquence d'initialisation du SPI UCBx

- Séquence d'initialisation du module UCBx pour le mode SPI :
  - Positionner à 1 (*set*) le bit UCSWRST du registre UCBx_clt1(software reset enable) :
    UCB0CTL1 |= UCSWRST ;
  - Intialiser les registres de l'UCBx
  - configurer les ports concernés
  - Positionner à 0 (*clear*) le bit UCSWRST du registre UCBx_clt1(software reset enable) :
    UCB0CTL1 &= ~UCSWRST ;
  - Facultatif : Autoriser les interruptions (TX ou RX) registre IE2
- Le mode SPI peut transmettre les charactères en 7 bits ou 8 bits (bit UC7BIT dans le registre UCBxCTL0).
- On peut transmettre en mode LSB ou MSB (bit UCMSB), en mode 7 bit, le registre de reception (UCxRXBUF) est aligné sur les LSB (MSB=0)

SPI mode Master

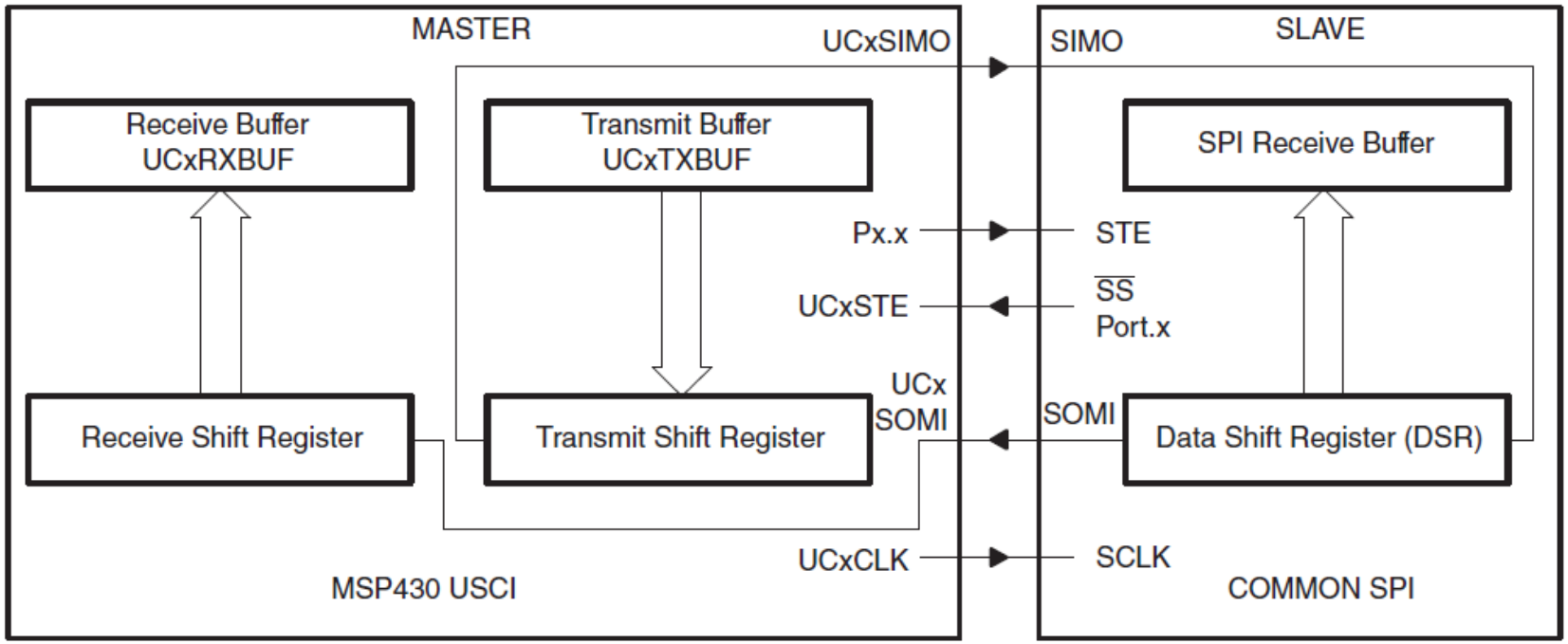


Figure 35-2. USCI Master and External Slave

Séquence d'émission/réception

- Il faut bien comprendre que les données vont dans les deux sens en parallèle :
  - du maître vers l'esclave sur SIMO
  - de l'esclave vers le maître sur SOMI
- La communication est initiée quand la donnée est mise dans le registre de transmission : UCBxTXBUF
- Le contenu de UCBxTXBUF est transmis au registre à décalage de transmission
- Le transfert bit par bit démarre sur UCBxSOMI et sur UCBxSIMO
- La reception de chaque bit est faite (sur l'esclave) sur le front opposé de l'horloge.
- Lorsque le charactère est reçu, le contenu du registre à décalage de réception est mis dans le registre de reception.
- l'interruption de reception est positionnée (UCBxRXIFG) indiquant que l'emission/reception est terminée.

SPI Timing

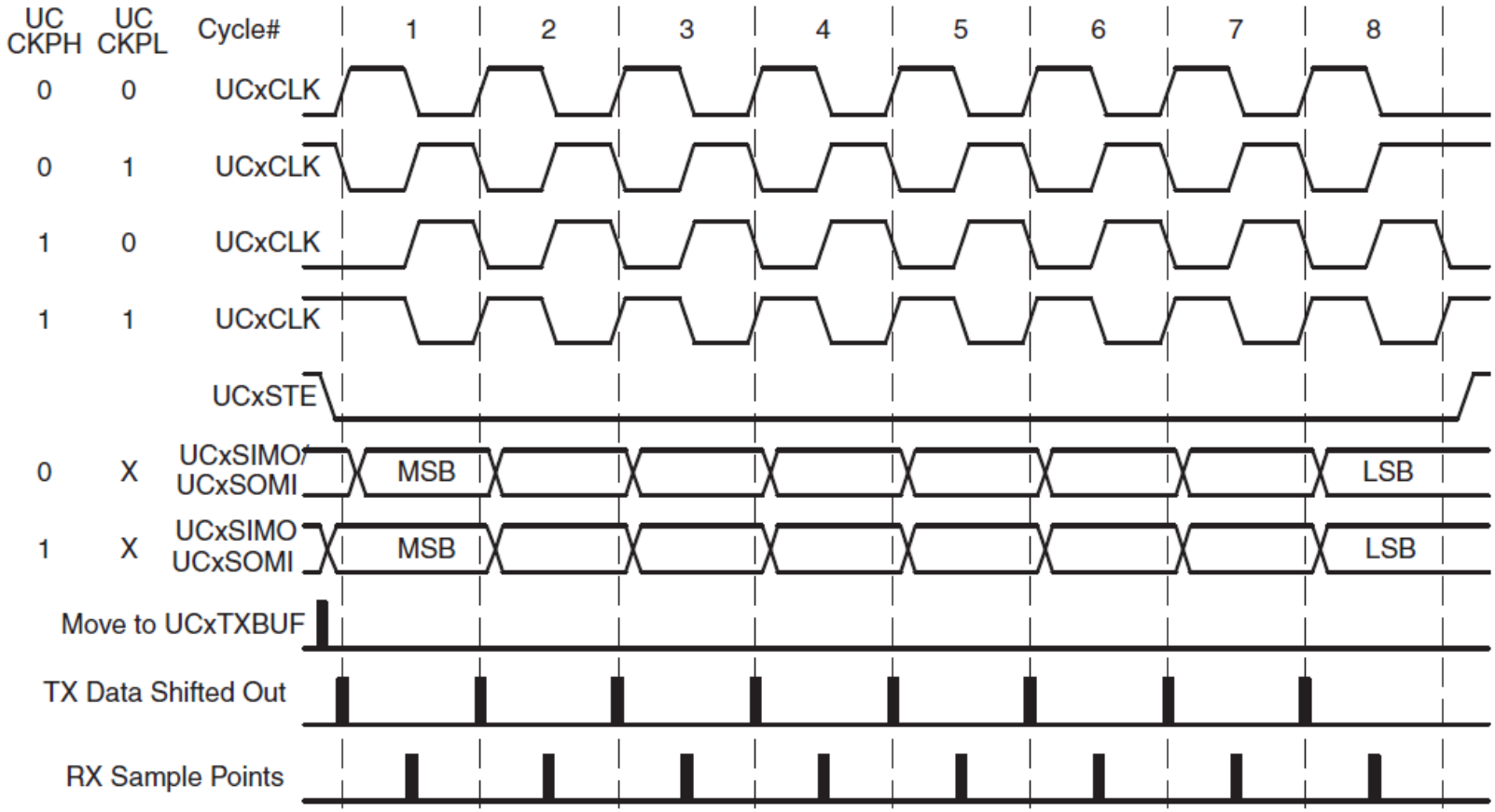


Figure 35-4. USCI SPI Timing With UCMSB = 1

- L'horloge est transmise par le maître sur UCBxCLK
- Sa fréquence est réglée (BRCLK/UCBRx)
- La phase et la polarité de transmission peuvent être réglées aussi

UCB0 Registers, SPI Mode

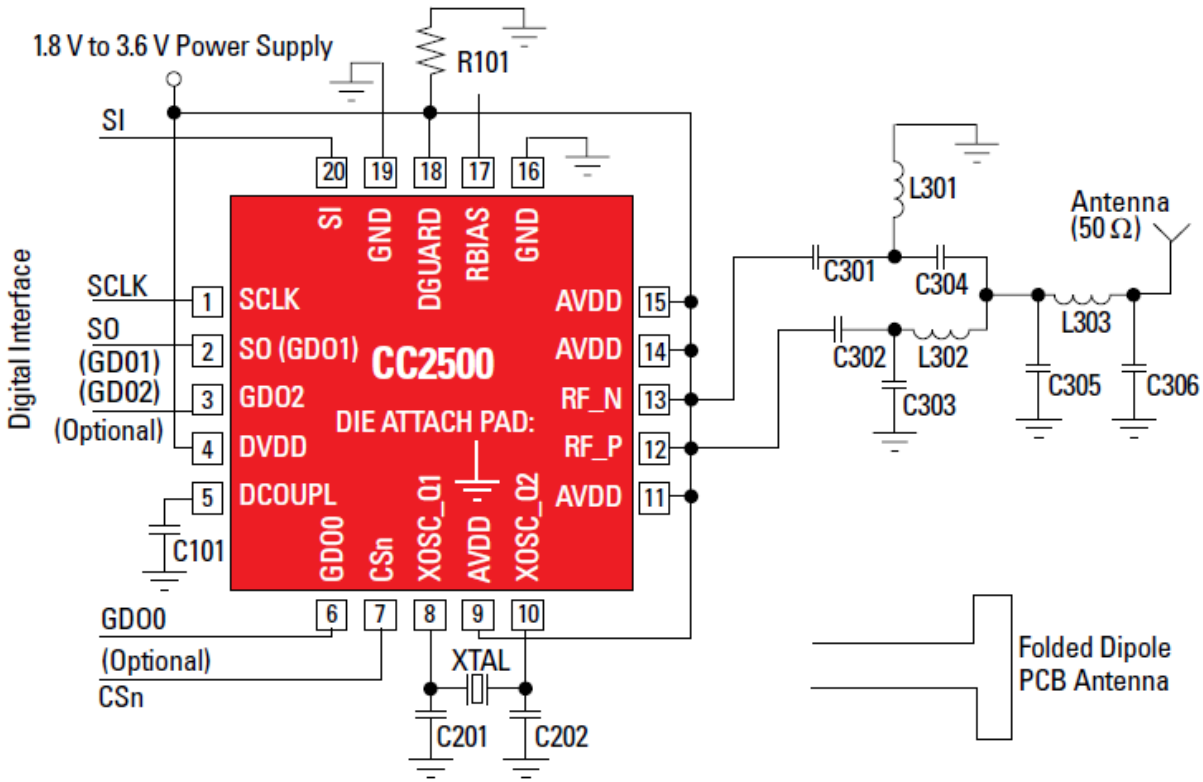
Table 35-14. USCI_B SPI Mode Registers

Offset	Acronym	Register Name	Type	Access	Reset	Section
00h	UCBxCTLW0	USCI_Bx Control Word 0	Read/write	Word	0101h	
00h	UCBxCTL1	USCI_Bx Control 1	Read/write	Byte	01h	Section 35.5.2
01h	UCBxCTL0	USCI_Bx Control 0	Read/write	Byte	01h	Section 35.5.1
06h	UCBxBRW	USCI_Bx Bit Rate Control Word	Read/write	Word	0000h	
06h	UCBxBR0	USCI_Bx Bit Rate Control 0	Read/write	Byte	00h	Section 35.5.3
07h	UCBxBR1	USCI_Bx Bit Rate Control 1	Read/write	Byte	00h	Section 35.5.4
08h	UCBxMCTL	USCI_Bx Modulation Control	Read/write	Byte	00h	Section 35.5.5
0Ah	UCBxSTAT	USCI_Bx Status	Read/write	Byte	00h	Section 35.5.6
0Bh		Reserved - reads zero	Read	Byte	00h	
0Ch	UCBxRXBUF	USCI_Bx Receive Buffer	Read/write	Byte	00h	Section 35.5.7
0Dh		Reserved - reads zero	Read	Byte	00h	
0Eh	UCBxTXBUF	USCI_Bx Transmit Buffer	Read/write	Byte	00h	Section 35.5.8
0Fh		Reserved - reads zero	Read	Byte	00h	
1Ch	UCBxICTL	USCI_Bx Interrupt Control	Read/write	Word	0200h	
1Ch	UCBxIE	USCI_Bx Interrupt Enable	Read/write	Byte	00h	Section 35.5.9
1Dh	UCBxIFG	USCI_Bx Interrupt Flag	Read/write	Byte	02h	Section 35.5.10
1Eh	UCBxIV	USCI_Bx Interrupt Vector	Read	Word	0000h	Section 35.5.11

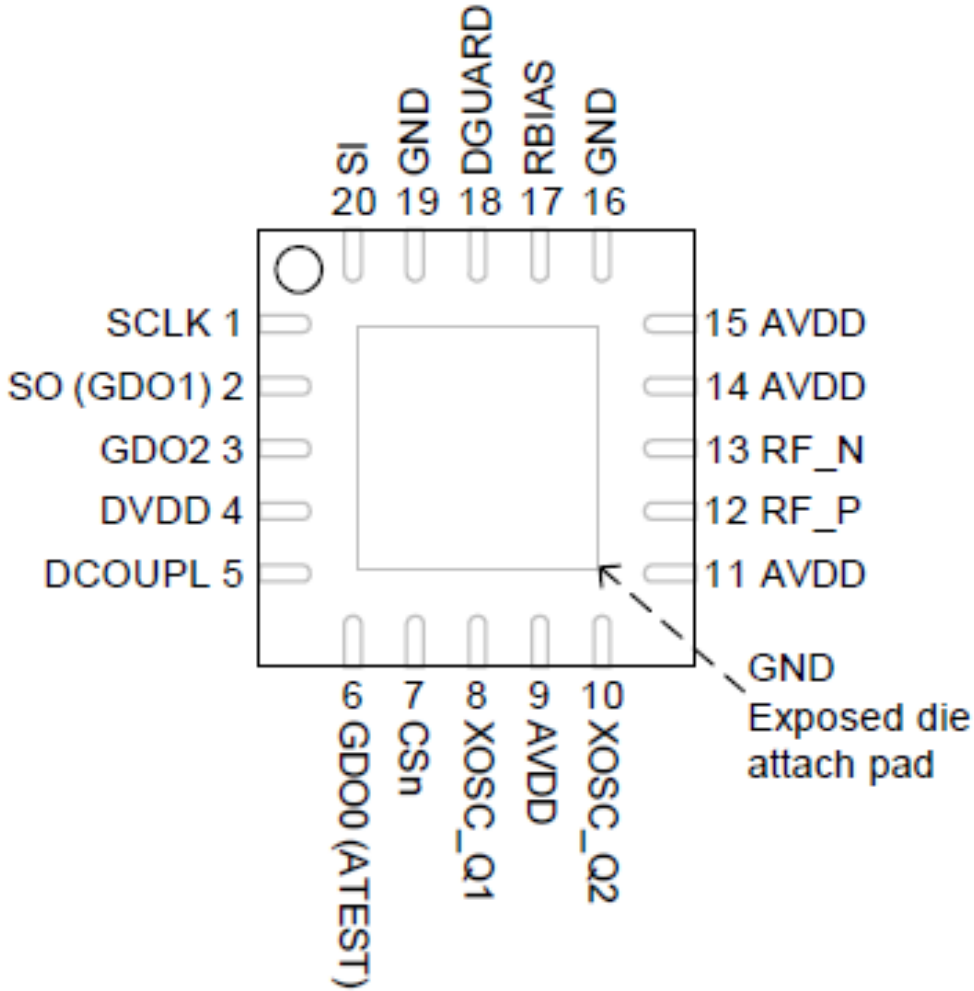
Introduction au CC2500

- Le CC2500 est un transceiver low power destiné à la bande 2.4-GHz (ISM : Industrial , Scientific and Medical), SRD (short range device) : 2400-2483.5 MHz
- Il n'a pas de couche MAC/PHY directement intégrée mais il est très paramétrable :
  - plusieurs modes de communication avec divers débits (jusqu'à 500 kbps),
  - plusieurs modulations,
  - optionnellement des codes correcteurs d'erreurs.
- Il est contrôlé par le MSP avec les 4 fils du SPI et propose aussi deux autre GPIO : GDO0 et GDO2 qui permettent de transmettre des interuptions (wake on radio par exemple)
- Il est intégré au sein de la carte CC2500EM qui est intégrable directement sur la carte EXP430 (connecteurs RF1, RF2)

Pinout du CC2500

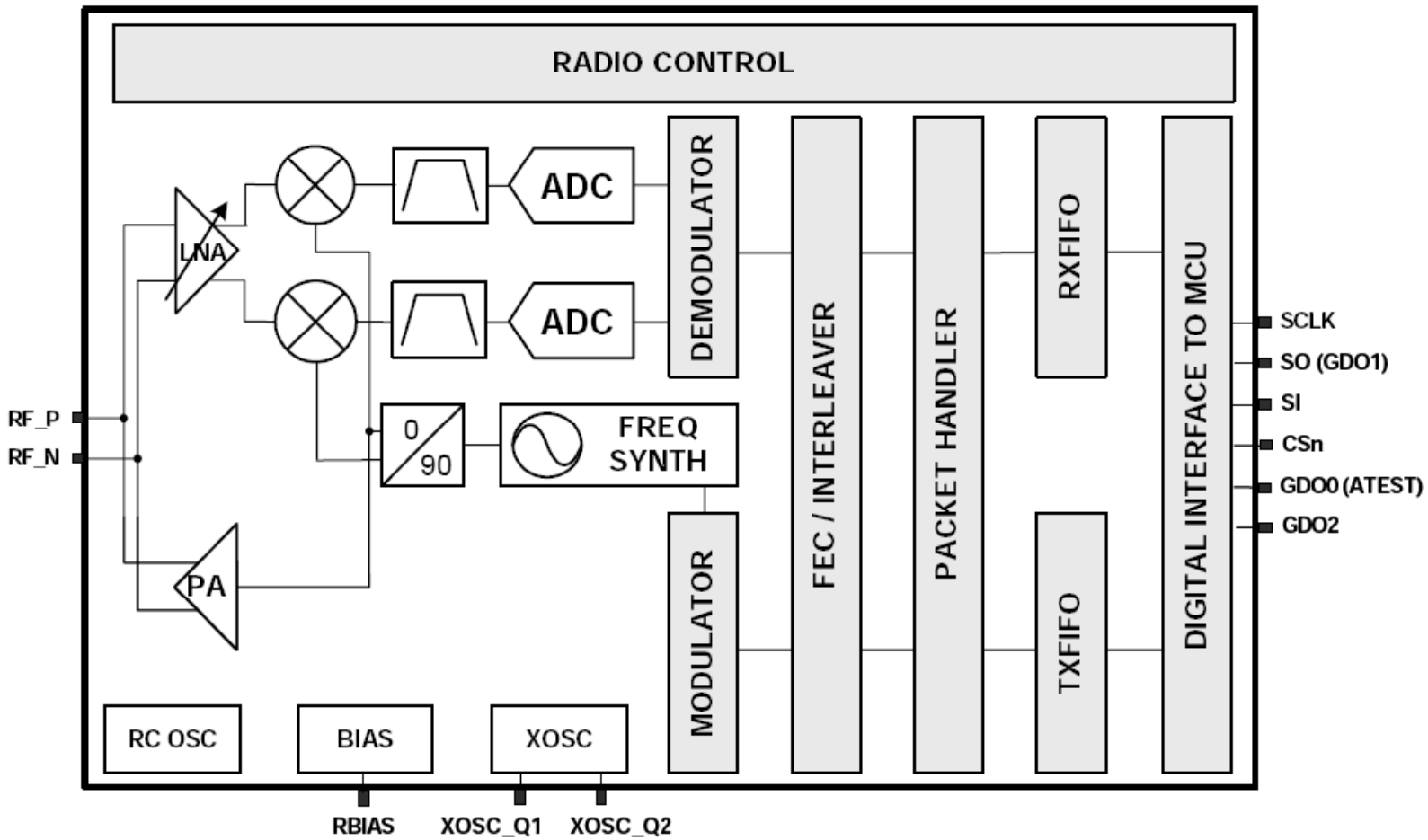


Pinout du CC2500 (2)



Source : Datasheet CC2500 (CC2500EMK.pdf)

Schéma du CC2500



États du CC2500

- Le CC2500 est une petite machine à états.
- Les états changent :
  - Soit suite à une commande envoyée du MSP
  - Soit suite à un évènement interne (envoie, reception de paquet)
- Exemple d'état : Idle, Fréquency synthesizer startup, receive mode, RX FIFO overflow etc..
- Le pilote logiciel (sur le MSP) du CC2500 doit connaître cette machine à état pour envoyer correctement les commandes de configuration.
- On peut aussi lire cet état dans le registre *status byte* qui est envoyé par le CC2500 au MSP à chaque communication SPI

Machine

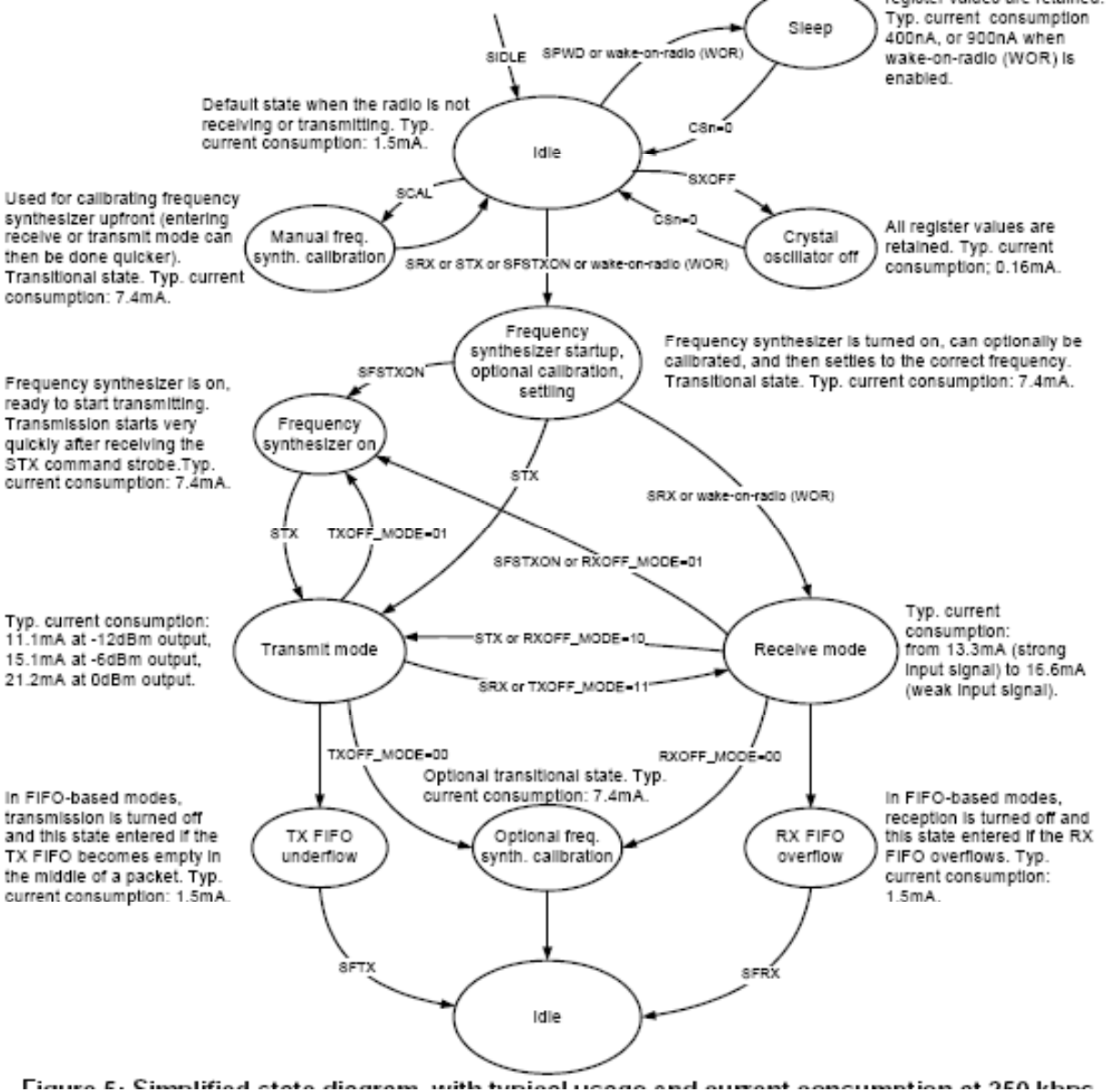


Figure 5: Simplified state diagram, with typical usage and current consumption at 250 kbps

Status Byte

Bits	Name	Description
7	CHIP_RDYn	Stays high until power and crystal have stabilized. Should always be low when using the SPI interface.
6:4	STATE[2:0]	Indicates the current main state machine mode
3:0	FIFO_BYTES_AVAILABLE[3:0]	The number of bytes available in the RX FIFO or free bytes in the TX FIFO (depends on the read/write-bit). If FIFO_BYTES_AVAILABLE=15, there are 15 or more bytes in RX FIFO or 49 or less bytes in the TX FIFO.

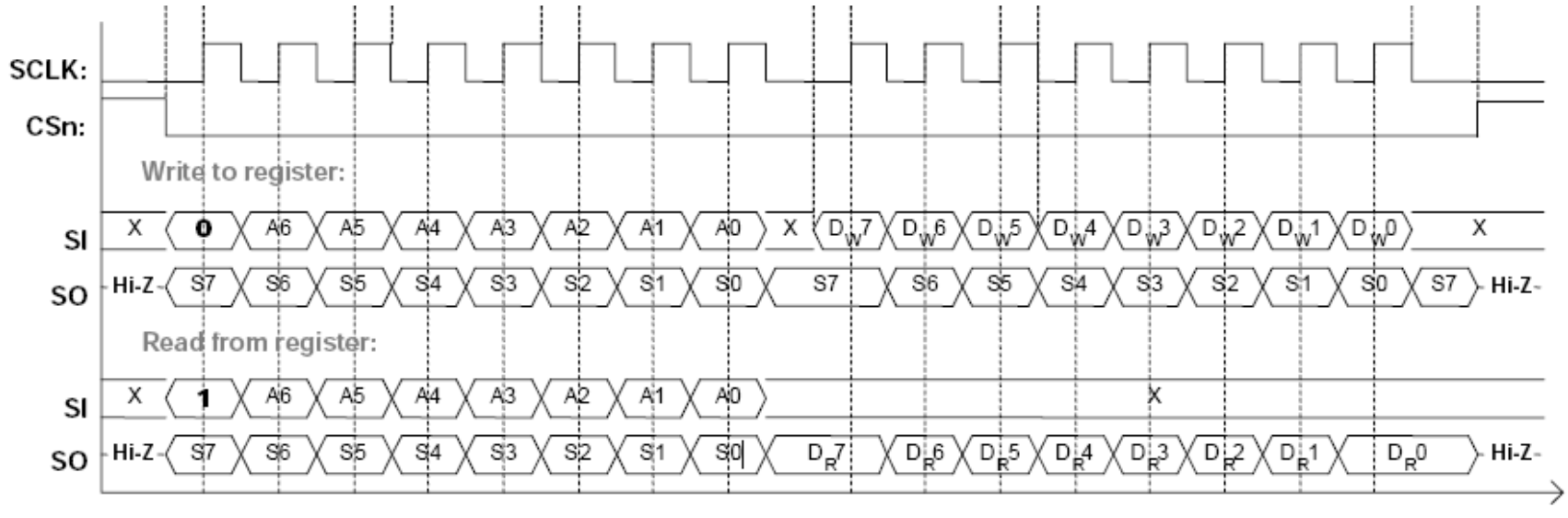
Value	State	Description
000	IDLE	Idle state (Also reported for some transitional states instead of SETTLING or CALIBRATE)
001	RX	Receive mode
010	TX	Transmit mode
011	FSTXON	Frequency synthesizer is on, ready to start transmitting
100	CALIBRATE	Frequency synthesizer calibration is running
101	SETTLING	PLL is settling
110	RXFIFO_OVERFLOW	RX FIFO has overflowed. Read out any useful data, then flush the FIFO with SFRX
111	TXFIFO_UNDERFLOW	TX FIFO has underflowed. Acknowledge with SFTX

Table 17: Status byte summary

Communication MSP-CC2500

- Le CC2500 est configuré grâce aux 4 fils de la connection SPI (SI, SO, SCLK et CSn), en tant qu'esclave.
- Toute transaction sur l'interface SPI commence par un octet d'en-tête qui contient :
  - le read/write bit (lecture ou écriture)
  - le burst access bit (accès en rafale)
  - l'adresse sur 6 bits (adresse interne au CC2500)
- Pendant le transfert le fil CSn (Chip Select, active low), doit être maintenu à 0.
- différent types d'accès aux registres du CC2500 :
  - Envoie d'une commande (Command Strobe)
  - Lecture/écriture d'un registre
  - Lecture/écriture de n registre (burst mode)
  - Lecture/écriture de n byte dans un des deux FIFOs (burst mode)

Programmation du CC2500 par le bus SPI



Attention aux timing...

Command Strobe

Address	Strobe Name	Description
0x30	SRES	Reset chip.
0x31	SFSTXON	Enable and calibrate frequency synthesizer (if MCSM0 . FS_AUTOCAL=1). If in RX (with CCA): Go to a wait state where only the synthesizer is running (for quick RX / TX turnaround).
0x32	SXOFF	Turn off crystal oscillator.
0x33	SCAL	Calibrate frequency synthesizer and turn it off (enables quick start). SCAL can be strobed in IDLE state without setting manual calibration mode (MCSM0 . FS_AUTOCAL=0)
0x34	SRX	Enable RX. Perform calibration first if coming from IDLE and MCSM0 . FS_AUTOCAL=1.
0x35	STX	In IDLE state: Enable TX. Perform calibration first if MCSM0 . FS_AUTOCAL=1. If in RX state and CCA is enabled: Only go to TX if channel is clear.
0x36	SIDLE	Exit RX / TX, turn off frequency synthesizer and exit Wake-On-Radio mode if applicable.
0x38	SWOR	Start automatic RX polling sequence (Wake-on-Radio) as described in Section 19.5.
0x39	SPWD	Enter power down mode when CSn goes high.
0x3A	SFRX	Flush the RX FIFO buffer. Only issue in IDLE, TXFIFO_UNDERFLOW or RXFIFO_OVERFLOW states.
0x3B	SFTX	Flush the TX FIFO buffer. Only issue in IDLE, TXFIFO_UNDERFLOW or RXFIFO_OVERFLOW states.
0x3C	SWORRST	Reset real time clock.
0x3D	SNOP	No operation. May be used to pad strobe commands to two bytes for simpler software.

Configuration Registers (extrait)

Address	Register	Description	Preserved in SLEEP state	Details on page number
0x00	IOCFG2	GDO2 output pin configuration	Yes	55
0x01	IOCFG1	GDO1 output pin configuration	Yes	55
0x02	IOCFG0	GDO0 output pin configuration	Yes	55
0x03	FIFOTHR	RX FIFO and TX FIFO thresholds	Yes	56
0x04	SYNC1	Sync word, high byte	Yes	56
0x05	SYNC0	Sync word, low byte	Yes	56
0x06	PKTLEN	Packet length	Yes	56
0x07	PKTCTRL1	Packet automation control	Yes	57
0x08	PKTCTRL0	Packet automation control	Yes	58
0x09	ADDR	Device address	Yes	58
0x0A	CHANNR	Channel number	Yes	58
0x0B	FSCTRL1	Frequency synthesizer control	Yes	59
0x0C	FSCTRL0	Frequency synthesizer control	Yes	59
0x0D	FREQ2	Frequency control word, high byte	Yes	59
0x0E	FREQ1	Frequency control word, middle byte	Yes	59
0x0F	FREQ0	Frequency control word, low byte	Yes	59
0x10	MDMCFG4	Modem configuration	Yes	60
0x11	MDMCFG3	Modem configuration	Yes	60
0x12	MDMCFG2	Modem configuration	Yes	61
0x13	MDMCFG1	Modem configuration	Yes	62
0x14	MDMCFG0	Modem configuration	Yes	62
0x15	DEVIATN	Modem deviation setting	Yes	63
0x16	MCSM2	Main Radio Control State Machine configuration	Yes	64
0x17	MCSM1	Main Radio Control State Machine configuration	Yes	65
0x18	MCSM0	Main Radio Control State Machine configuration	Yes	66

Status Registers

Address	Register	Description	Details on page number
0x30 (0xF0)	PARTNUM	*CC2500* part number	75
0x31 (0xF1)	VERSION	Current version number	75
0x32 (0xF2)	FREQEST	Frequency Offset Estimate	75
0x33 (0xF3)	LQI	Demodulator estimate for Link Quality	76
0x34 (0xF4)	RSSI	Received signal strength indication	76
0x35 (0xF5)	MARCSTATE	Control state machine state	76
0x36 (0xF6)	WORTIME1	High byte of WOR timer	77
0x37 (0xF7)	WORTIME0	Low byte of WOR timer	77
0x38 (0xF8)	PKTSTATUS	Current GDOx status and packet status	77
0x39 (0xF9)	VCO_VC_DAC	Current setting from PLL calibration module	77
0x3A (0xFA)	TXBYTES	Underflow and number of bytes in the TX FIFO	77
0x3B (0xFB)	RXBYTES	Overflow and number of bytes in the RX FIFO	78

Résumé du contrôle du CC2500

- 14 commandes (p. 51), les commandes sont éxécutées immédiatement sauf SPWD et SXOFF qui sont éxecutées quand CSn passe à 1.
- 48 registres de configuration (p. 52)
- 12 registres d'états
- 2 FIFOs de 64 octets à l'adresse 0x3F, en écriture ou en lecture suivant l'opération.
- l'adresse d'un registre est codée sur 6 bits, les quatres bits suivant sont utilisé pour distinguer R/W et burst/byte (+0x00,+0x40,+0x80,+0xC0)
- Exemple de valeurs de l'octet d'en-tête pour accéder à une FIFO :
  - 0x3F : simple accès à la TX-FIFO
  - 0x7F : Accès Burst à la TX-FIFO
  - 0xBF : simple accès à la RX-FIFO
  - 0xFF : Accès Burst à la RX-FIFO

Configuration de la radio

- Data Rate : (MDMCFG3.DRATE_M et DRATE_E) :

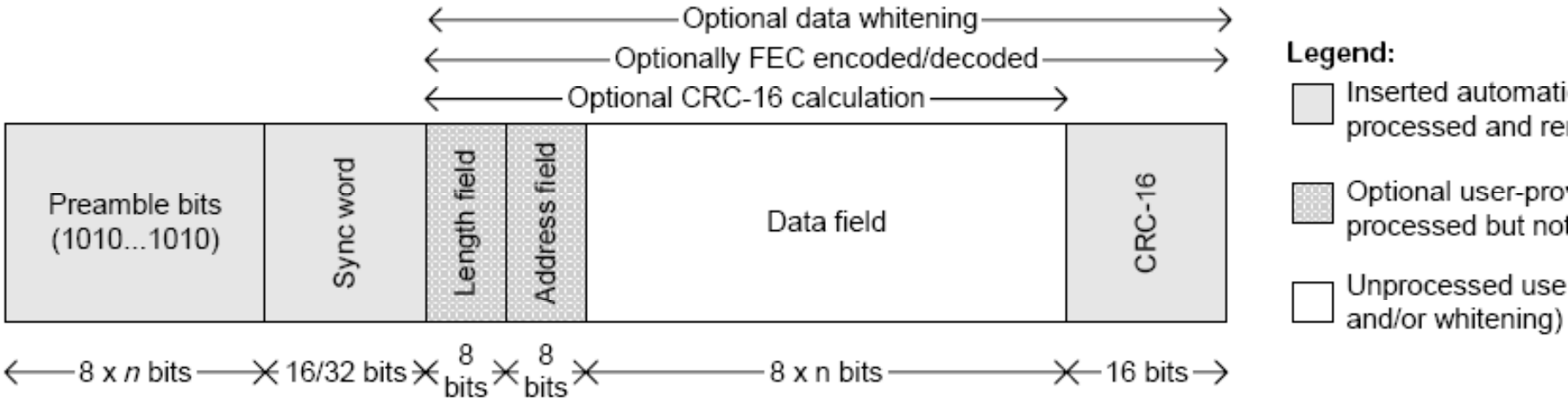
$$R_{data} = \frac{(256 + DRATE\_M).2^{DRATE\_E}}{2^{28}}.f_{osc}$$

- Largeur de bande du canal en reception (MDMCFG4.CHANBW_E et CHANBW_M) :

$$BW_{chan} = \frac{f_{osc}}{8.(4 + CHANBW\_M).2^{CHANBW\_E}}$$

- Frequency Offset Compensation : pour les modulation FSK, GFSK ou MSK le registre d'état FREQEST peut être utilisé pour ajuster la fréquence de l'oscillateur.
- Bit synchronization
- Byte synchronisation : Sync Word

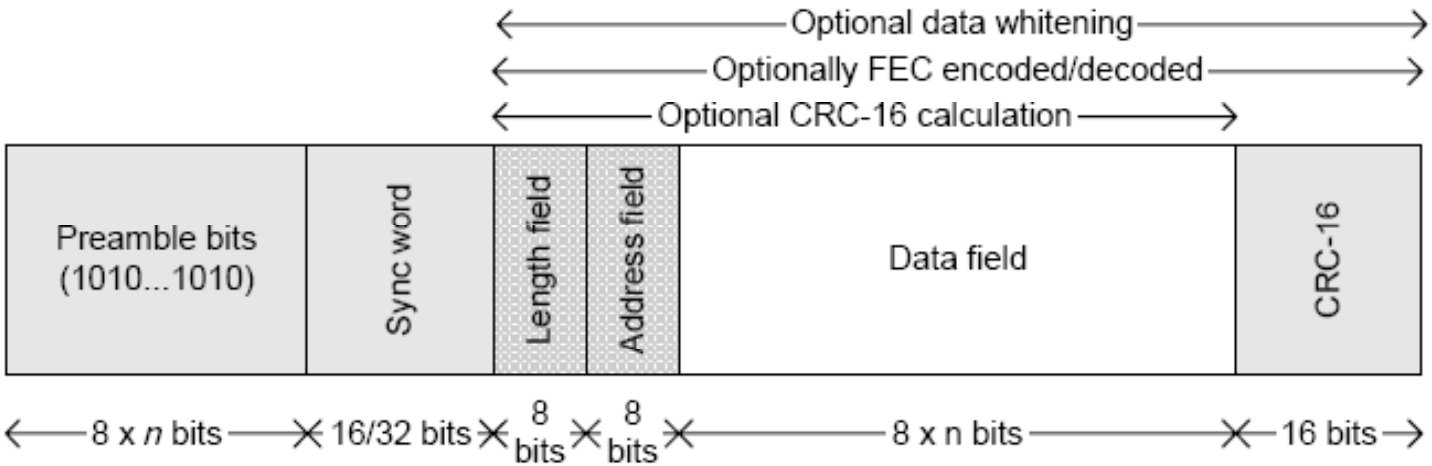
format du paquet du CC2500



mode transmission on ajoute :

- Un nombre programmable d'octet de préambule (en général 4)
- un ou deux mot de synchronisation (généralement 4 octets)
- Un octet de longueur
- Un octet d'adresse
- Optionnellement un checksum sur le champs data.
- Optionnellement entrelacement et code correcteur
- Optionnellement Une décorrelation du paquet

format du paquet du CC2500



reception effectue :

- détection de préambule
- Synchronisation au niveau mot
- vérification de l'adresse (optionnel)
- verification CRC (optionnel)
- rajoute optionnellement les deux registres : Received packet status byte avec le RSSI, CRC et LQI.

Contrôle de la radio

- Le diagramme complet de la machine à état du CC2500 est représenté en figure 15 p. 35.
- Les transitions ne sont pas instantannées (presque une milliseconde pour calibrer et passer en réception)

Description	XOSC periods	26 MHz crystal
IDLE to RX, no calibration	2298	88.4 µs
IDLE to RX, with calibration	~21037	809 µs
IDLE to TX/FSTXON, no calibration	2298	88.4 µs
IDLE to TX/FSTXON, with calibration	~21037	809 µs
TX to RX switch	560	21.5 µs
RX to TX switch	250	9.6 µs
RX or TX to IDLE, no calibration	2	0.1 µs
RX or TX to IDLE, with calibration	~18739	721 µs
Manual calibration	~18739	721 µs

Table 28: State transition timing

Autre paramêtres du CC2500

- Threshold pour les FIFO (seuil)
- Puissance su signal de sortie
- selectivité en reception
- GPIO : GDO0 et GDO2, peuvent être drivé par différent signaux en fonction du registre GDOx_CFG (table 33 p. 47)

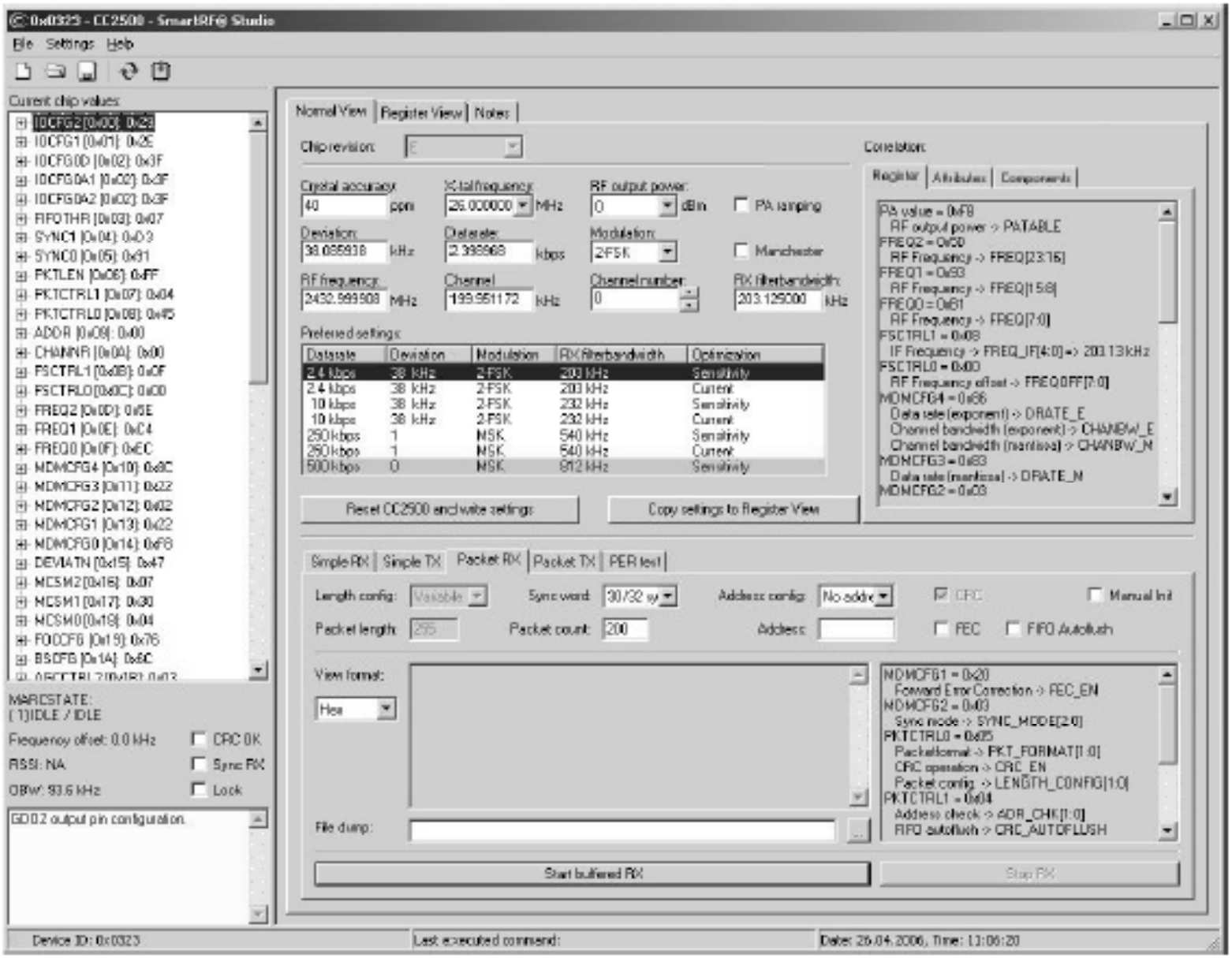
Écriture de pilotes

L'écriture d'un pilote doit suivre les principes suivants :

- Contrôle de la machine à état depuis le logiciel.
- Utilisation de toutes les fonctionnalités matérielles.
- Fournir une abstraction aux autres parties de l'application.
- Pour la configuration de la radio on utilise l'outils SmartRF fournit par texas.

Envoyer un paquet sur la radio ne doit pas être plus compliqué qu'un appel de fonction.

SmartRF


0x0323 - CC2500 - SmartRF® Studio
File Settings Help
Current chip values:
Normal View
Register View
Notes
Chip revision:
Correlation:
Crystal accuracy:
40
ppm
Xtal frequency:
26.000000
MHz
RF output power:
0
dBm
PA ramping
Deviation:
38.085938
kHz
Datarate:
2.399963
kbps
Modulation:
2-FSK
Manchester
RF frequency:
2432.999908
MHz
Channel:
199.951172
kHz
Channel number:
0
RX filterbandwidth:
203.125000
kHz
Preferred settings:
Datarate
Deviation
Modulation
RX filterbandwidth
Optimization
2.4 kbps
38 kHz
2-FSK
203 kHz
Sensitivity
2.4 kbps
38 kHz
2-FSK
203 kHz
Current
10 kbps
38 kHz
2-FSK
232 kHz
Sensitivity
10 kbps
38 kHz
2-FSK
232 kHz
Current
250 kbps
1
MSK
540 kHz
Sensitivity
250 kbps
1
MSK
540 kHz
Current
500 kbps
0
MSK
812 kHz
Sensitivity
Reset CC2500 and write settings
Copy settings to Register View
Simple RX
Simple TX
Packet RX
Packet TX
PER test
Length config:
Variable
Sync word:
30/32 sy
Address config:
No addr
CRC
Manual Init
Packet length:
255
Packet count:
200
Address:
FEC
FIFO Autoflush
View format:
Hex
File dump:
Start buffered RX
Stop RX
MARCSTATE:
(1) IDLE / IDLE
Frequency offset: 0.0 kHz
CRC OK
RSSI: N/A
Sync RX
OBW: 93.6 kHz
Lock
GDO2 output pin configuration:
Device ID: 0x0323
Last executed command:
Date: 26.04.2006, Time: 13:06:20

Pilote d'émission en attente

```
/* pkt < 64 bytes, wait EOP */
void cc1100_utx(char *buffer, uint8_t length)
{
  cc1100_tx_error = 0;
  cc1100_check_fifo_xflow_flush();
  /* Fill tx fifo */
  CC1100_SPI_TX_FIFO_BYTE  (length);
  CC1100_SPI_TX_FIFO_BURST (buffer, length);
  /* Send packet and wait for complete */
  cc1100_gdo2_set_signal(CC1100_GDOx_SYNC_WORD);
  CC1100_HW_GDO2_DINT();
  CC1100_SPI_STROBE(CC1100_STROBE_STX);
  while (! ( CC1100_HW_GDO2_READ() )); /* GDO2 goes high = SYNC TX
*/
  while (  ( CC1100_HW_GDO2_READ() )); /* GDO2 goes low   = EOP
*/
  CC1100_HW_GDO2_EINT();
}
```

Pilote d'émission avec intr. (1)

```
void cc1100_tx(char *buffer, uint8_t length)
{
  uint8_t txbytes; /* bytes free in the fifo */
  uint8_t tosend;

  cc1100_check_fifo_xflow_flush();
  cc1100_tx_packet  = buffer;
  cc1100_tx_length  = length;
  cc1100_tx_offset  = 0;
  cc1100_tx_ongoing = 1;
  cc1100_tx_error   = 0;
  cc1100_tx_sent    = 0;
  /* Fill tx fifo */
  CC1100_SPI_TX_FIFO_BYTE  (length);
  TX_WRITE_BLOCK();

  /* Send packet but don't wait complete */
  CC1100_HW_GDO2_DINT();
  CC1100_HW_GDO0_DINT();
  cc1100_gdo0_set_signal(CC1100_GDOx_TX_FIFO);    /* gdo0 tx fifo              */
  cc1100_gdo2_set_signal(CC1100_GDOx_SYNC_WORD); /* gdo2 sync & eop            */
  CC1100_HW_GDO2_IRQ_ON_DEASSERT();              /* want an irq for EOP        */
  CC1100_HW_GDO0_IRQ_ON_DEASSERT();              /* want an irq on Tx < thr */
  CC1100_SPI_STROBE(CC1100_STROBE_STX);          /* start                      */
  while (! ( CC1100_HW_GDO2_READ() ));          /* GDO2 high = sync TX        */
  CC1100_HW_GDO0_EINT();                         /* allow irq on gdo0          */
  CC1100_HW_GDO2_EINT();                         /* allow irq on gdo2          */
}
```

Pilote d'émission avec intr. (2)

```
void cc1100_tx_pkt_data()
{
  uint8_t txbytes; /* bytes free in the fifo */
  uint8_t tosend;

  CC1100_SPI_ROREG(CC1100_REG_TXBYTES, txbytes);
  txbytes = 63 - txbytes; /* room free in Tx FIFO */
  tosend = MIN(txbytes, cc1100_tx_length);
  CC1100_SPI_TX_FIFO_BURST(cc1100_tx_packet, tosend);
  cc1100_tx_packet += tosend;
  cc1100_tx_length -= tosend;
}


/* called when done on asynchroneous Tx */
void cc1100_tx_done_intr(void)
{
  if (cc1100_check_fifo_xflow_flush())
    {
      cc1100_tx_error = 1;
    }
  cc1100_tx_ongoing = 0;
  cc1100_tx_sent    = 1;

  CC1100_HW_IRQ_PACKET_ASSERT();
}
```

Pilote d'émission avec intr. (3)

```
void cc1100_interrupt_handler(uint8_t pin)
{
  int interrupt_policy;
  if (pin == CC1100_GDO0)
    interrupt_policy = cc1100_gdo0_cfg;
  else if (pin == CC1100_GDO2)
    interrupt_policy = cc1100_gdo2_cfg;
  else
    return;

  switch (interrupt_policy)
    {
    case CC1100_GDOx_TX_FIFO:      /* Tx FIFO threshold */
      cc1100_tx_pkt_data();
      break;
    case CC1100_GDOx_RX_FIFO:      /* Rx FIFO threshold */
      cc1100_rx_pkt_data();
      break;
    case CC1100_GDOx_SYNC_WORD:
      if (cc1100_tx_ongoing == 1) /* Tx EOP */
        cc1100_tx_done_intr();
      else if (cc1100_rx_ongoing) /* Rx EOP */
        cc1100_rx_pkt_data();
      else                        /* Rx SYNC */
        cc1100_rx_pkt_start();
      break;
    case CC1100_GDOx_CHIP_RDY:
      break;
    default:
      break;
    }
}
```

Ecriture d'une couche MAC/Réseau

- CSMA (plutôt que TDMA)
- Adresse des noeuds, table de routage
- protocole complet etc..