LORRAINE INP Ensem

NRJ 3A – B18

PHIL

Partie 2 :

Delphino 28379D

Lotfi BAGHLI

Lotfi.Baghli@univ-lorraine.fr

2022

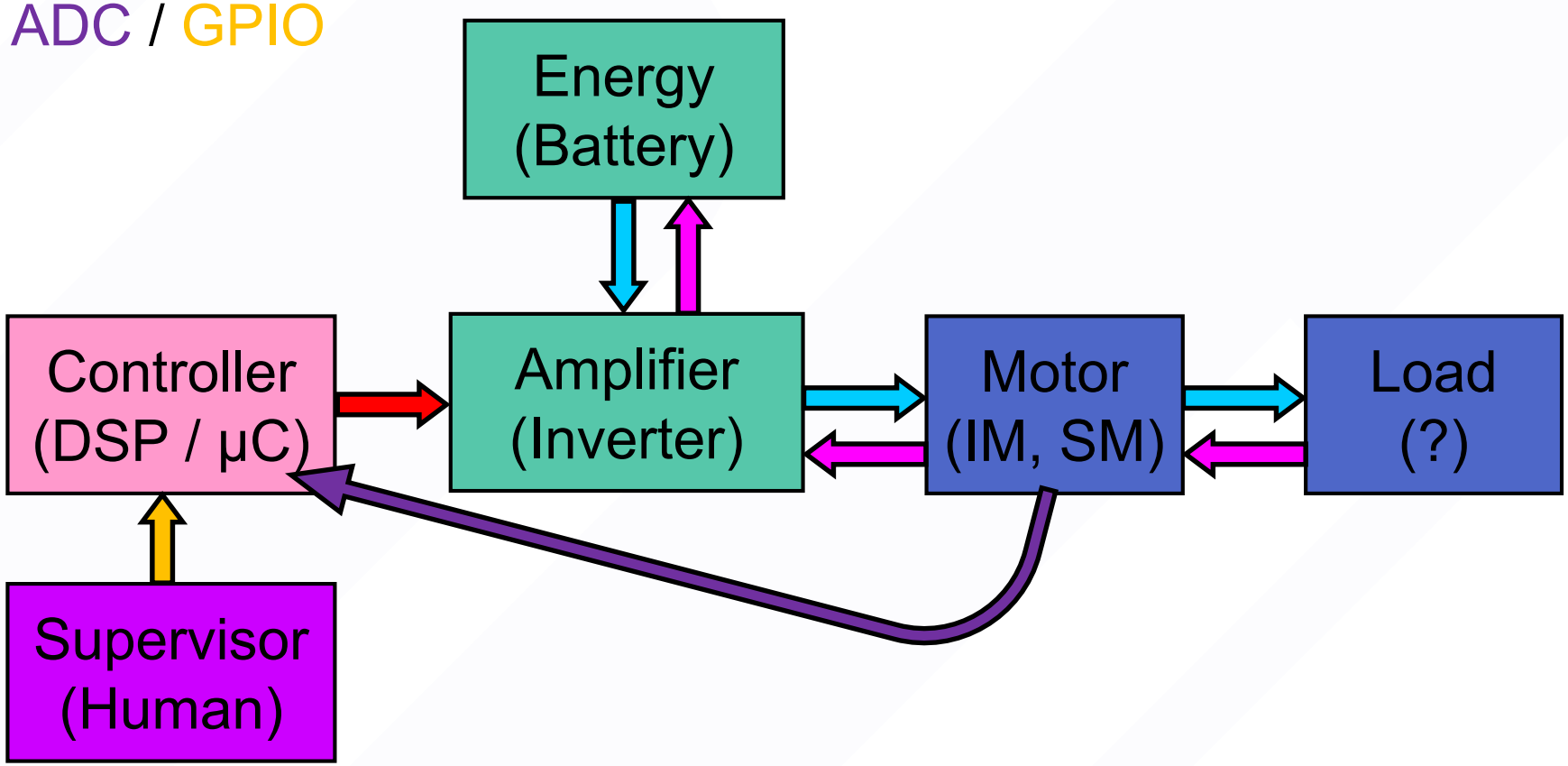
Plan de la partie 2 : Périphériques

- Launchpad, DSC F28379D et ses périphériques
- Entrées/sorties : GPIO
- Interruptions et ISR
- Modulation de Largeur d'Impulsion : PWM
- Conversion Analogique Numérique : ADC

- Cours sur Arche https://arche.univ-lorraine.fr/user/index.php?id=51101
- TMS320F2837xD Technical Reference Manual spruhm8i sur Arche: https://arche.univ-lorraine.fr/pluginfile.php/2931136/mod_folder/content/0/TMS320F2837xD%20Technical%20Reference%20Manual%20spruhm8i.pdf?forcedownload=1

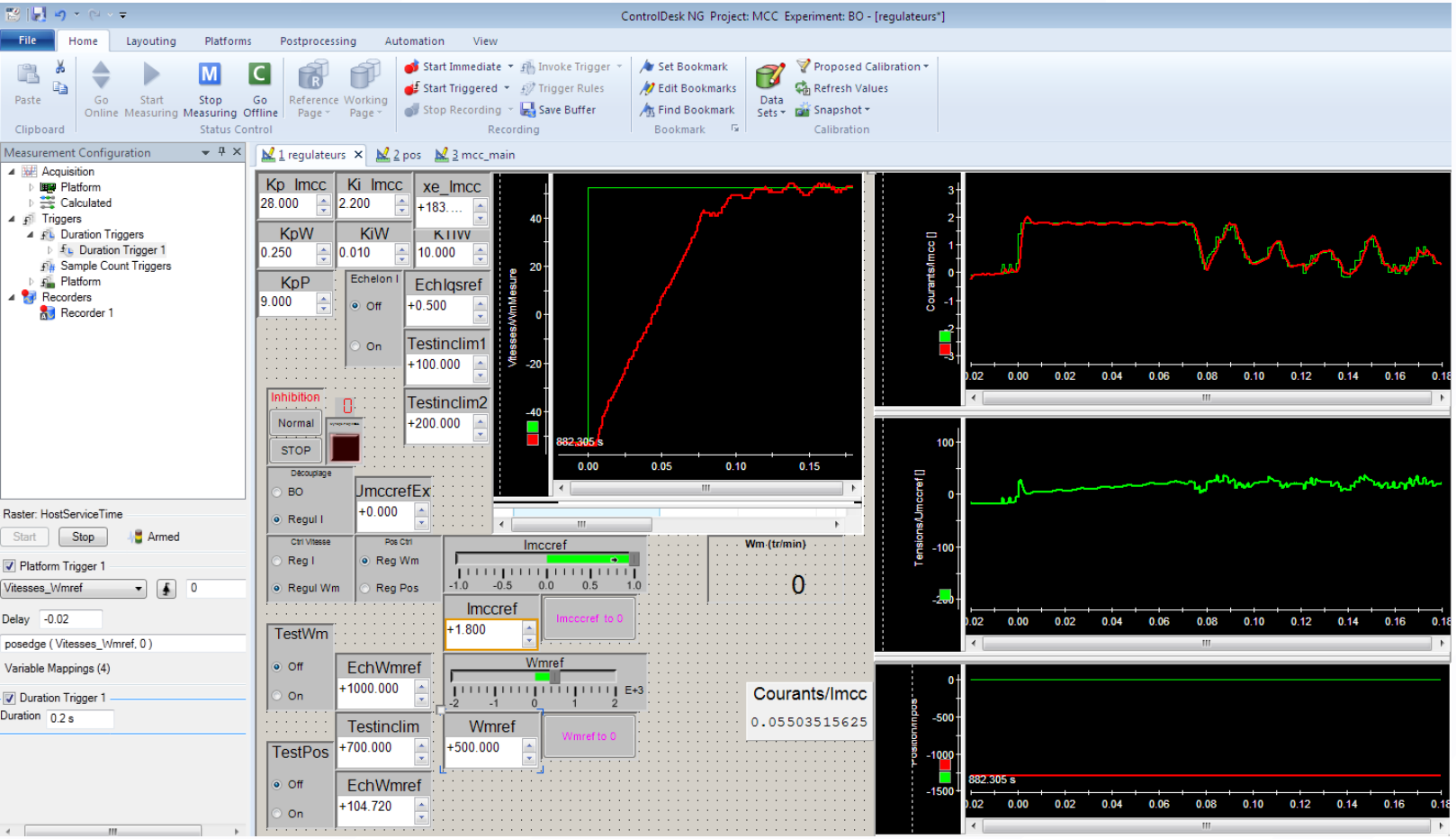
Introduction

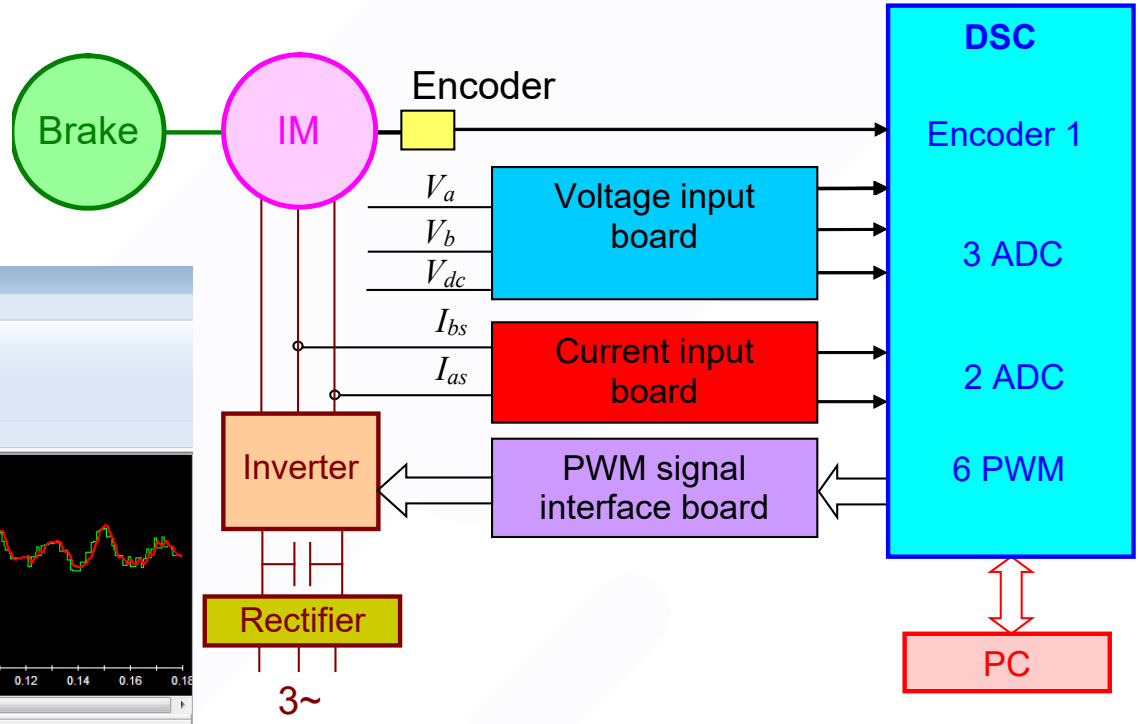
- Exemple de contrôle d'un moteur
- PWM / ADC / GPIO

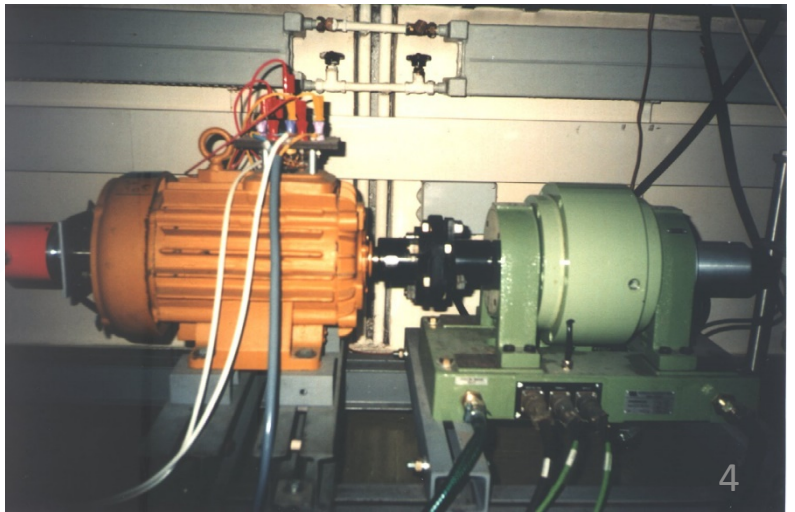


Introduction

- Exemple de contrôle d'un moteur
- Boucles de courant / vitesse







Solutions for rapid prototyping

Choice criteria

- Test board and components prices
- Performances : 8/16/32/64 bits, MHz, FPU, DSP
- Development tools: free, easy to use, debugging, HMI
- Simulation and rapid prototyping tools: Tinkercad, Fritzing, Altium, Eagle, Proteus
- Popularity: forums, tutorials, user's guides, application notes
- Usage categories
  - Initiation to microelectronics (µC)
  - Student projects / college: robots, interactions,…
  - Master and research in EE: Machine control, Power electronics, communication, IoT

https://www.tinkercad.com
https://fritzing.org

Solutions for rapid prototyping

Components

- 8 bits : AVR, PIC 18F, MSP430
- 16 bits : PIC 24F, dspic 30F, 33F, 33E
- 32 bits : PIC 32, Piccolo F28027F / F28069M, Delfino F283xxx, STM32, Cortex M3, M4
- Requires circuit board
- Connector et programmer (ICSP, JTAG, ISP AVR : 5 to 500 USD)
- Quartz+2C, PB, LED, Connecters, Pads
- Optimal solution / hit market: Size and cost

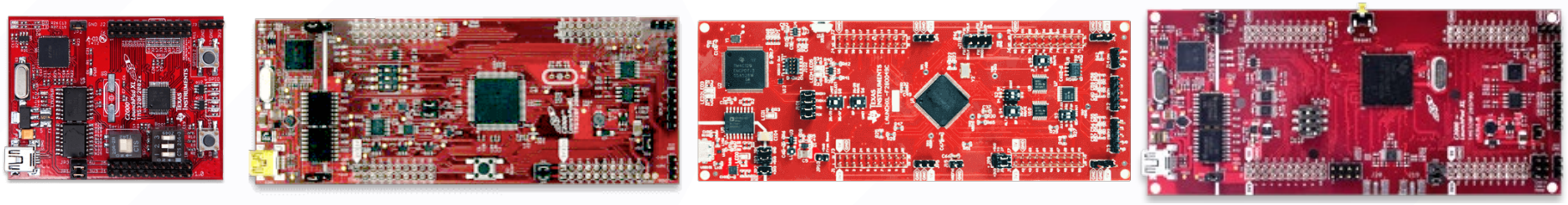
Development boards

- 8 bits : Arduino Uno, nano, Lily pad, LaunchPad MSP430, Mplab Xpress PIC18855
- 16 bits : Curiosity PIC 24F, dspic 30F, 33F, 33E
- 32 bits : Teensy 3.2, Node MCU, ESP8266, ESP32, Wemos, MBed, STM32 nucleo, LaunchePad C2000
- 64 bits : Raspberry Pi3, Pi zero
- Best price: 3 to 45 USD
- Ready to use, rapid prototyping, Boot loader, USB
- Format DIL (Mbed, STM, Arduino nano, Teensy) nucleo, Launch XL, Arduino uno
- Shields : LCD display, OLED, 7-seg., Wifi, BT, DHT, Motor drivers, light, customize

Solutions for rapid prototyping

LaunchPad C2000

http://www.ti.com/lsds/ti/tools-software/launchpads/launchpads.page#performance



LAUNCHXL-F28027F

Real time MCU with InstaSPIN-FOC motor control software.

- Featuring the 32-bit C2000 TMS320F28027F MCU @ **60MHz**, 64kB Flash / 12kB RAM. InstaSPIN-FOC enabled.
- Includes 8 PWM channels (4 high resolution), 12-bit 4.6 MSPS ADC, 3x 32-bit timers, temperature sensor, comparator.
- Motor control software in ROM implements InstaSPIN-FOC solutions for 3 phase motors.

17.00 USD

LAUNCHXL-F28069M

High performance controller with InstaSPIN motor control software.

- Featuring the 32-bit C2000 TMS320F28069M MCU @ **90MHz**, **FPU,** 256kB Flash / 96kB RAM. InstaSPIN-MOTION and InstaSPIN-FOC enabled.
- Includes 16 PWM channels (8 high resolution), 16 channel 12-bit 4.6 MSPS ADC, 3x 32-bit timers, 6 channel DMA, temperature sensor.
- Motor control software in ROM implements InstaSPIN-FOC and InstaSPIN-MOTION solutions for 3 phase motors.

24.99 USD

LAUNCHXL-F28049C

Optimized for cost-sensitive power control applications.

- Featuring the TMS320F280049C: 32-bit C2000 MCU @ **100MHz, FPU**, TMU, and CLA; 256kB Flash / 100kB RAM; InstaSPIN-FOC and Configurable Logic Block enabled.
- Includes 16 PWM channels (16 high resolution), 21 channel 3x12-bit 3.46 MSPS ADC, 14 Comparators, 4 Sigma Delta Filters, 2x CAN
- Motor control software in ROM implements InstaSPIN-FOC solutions for 3 phase motors.

30.00 USD

new

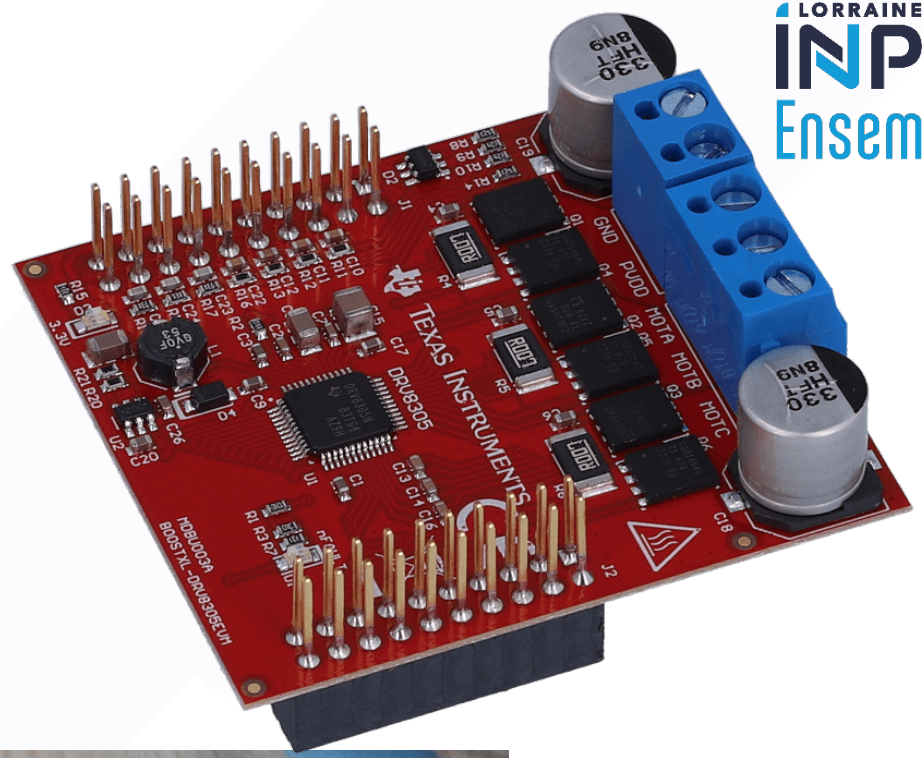
LAUNCHXL-F28379D

Dual Core MCU with highest level of peripheral integration in C2000 product family.

- Featuring the TMS320F28379D: **dual** 32-bit **Delfino** MCUs @**200MHz**, **FPU,** 1MB Flash/164KB RAM. Position Manager Enabled.
- 4 16/12-bit ADCs with on PCB instrumentational amplifiers. 3x 12-bit output DACs and 8 windowed comparators for asychronous output triggering.
- 24 PWMs (16 high resolution), 6 eCAP modules mapped to any GPIO,and 3 eQEP modules. 8 Sigma Delta Demodulator Inputs.

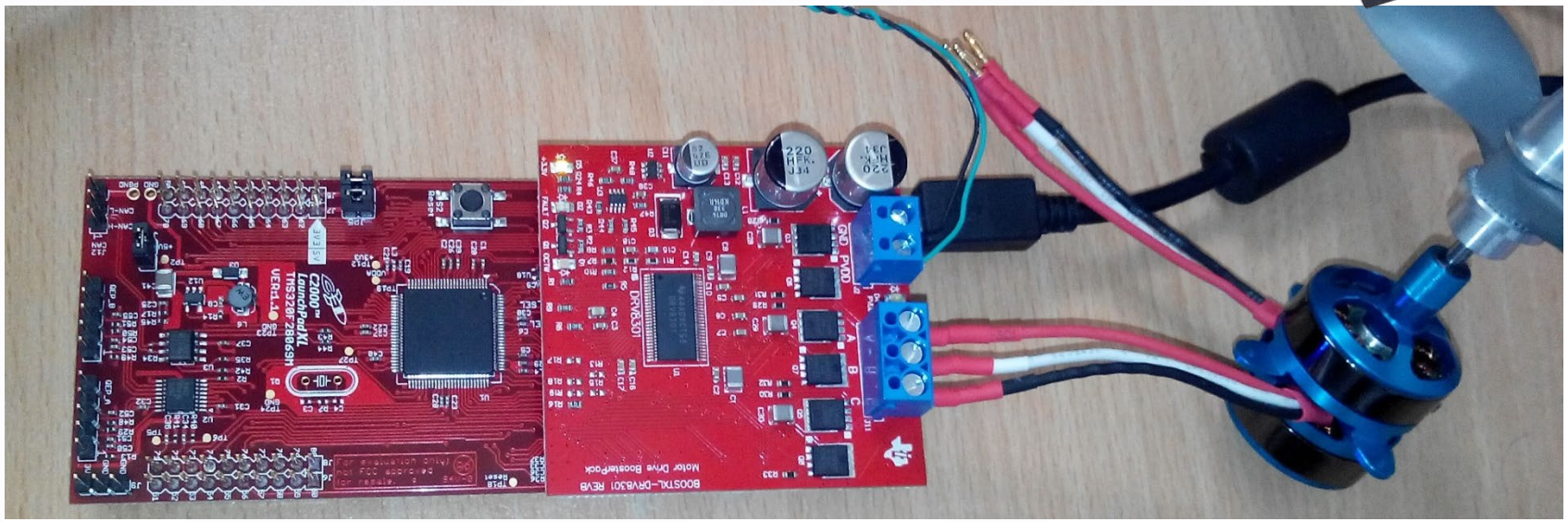
33.79 USD

Solutions for rapid prototyping



LaunchPad C2000

- LaunchXL-F28379D Quick Start Guide SPRUI40.pdf
- LaunchXL-F28379D User's guide SPRUI77C.pdf
- BoosterPack plug-in modules
- BOOSTXL-DRV8301 : tri-phase inverter 10A, 6-24V
- BOOSTXL-DRV8305EVM: tri-phase inverter 15A, 48V



https://www.ti.com/tool/BOOSTXL-DRV8305EVM

Solutions for rapid prototyping

BoosterPacks

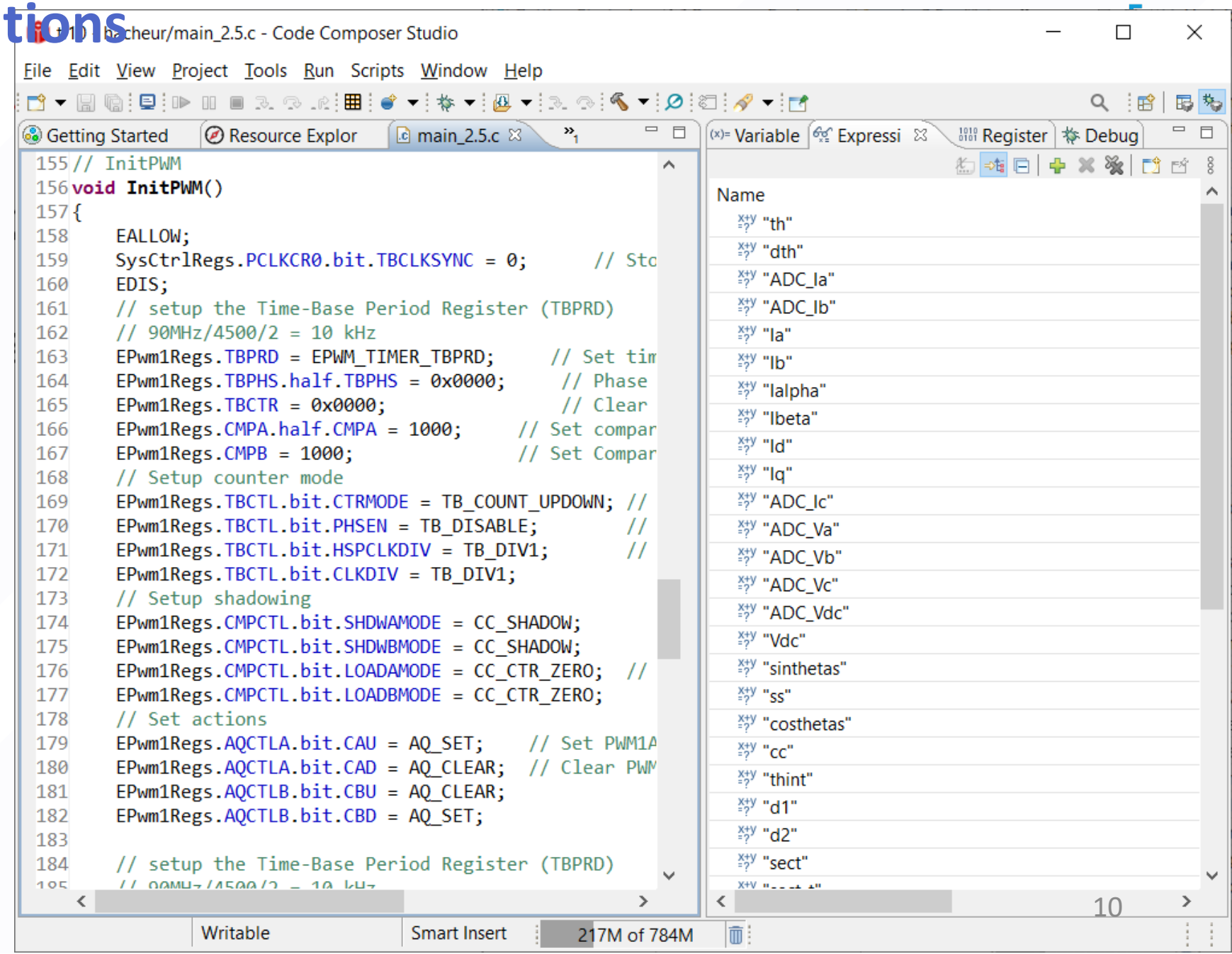
- Texas Instruments
- If Booster packs are compatible and stackable

https://www.ti.com/design-resources/embedded-development/hardware-kits-boards.html#boosterpack
https://www.ti.com/tool/BOOSTXL-SENSORS?keyMatch=BOOSTERPACK%20EVALUATION%20MODULE

Software Tools Solutions

Code Composer Studio

- Texas Instruments
- TMS320
- C/C++ Compiler
- Debugger




```
155 // InitPWM
156 void InitPWM()
157 {
158     EALLOW;
159     SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;      // Stc
160     EDIS;
161     // setup the Time-Base Period Register (TBPRD)
162     // 90MHz/4500/2 = 10 kHz
163     EPwm1Regs.TBPRD = EPWM_TIMER_TBPRD;     // Set tim
164     EPwm1Regs.TBPHS.half.TBPHS = 0x0000;     // Phase
165     EPwm1Regs.TBCTR = 0x0000;                // Clear
166     EPwm1Regs.CMPA.half.CMPA = 1000;     // Set compar
167     EPwm1Regs.CMPB = 1000;               // Set Compar
168     // Setup counter mode
169     EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; //
170     EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;        //
171     EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       //
172     EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
173     // Setup shadowing
174     EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
175     EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
176     EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;  //
177     EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
178     // Set actions
179     EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;    // Set PWM1A
180     EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;  // Clear PWM
181     EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
182     EPwm1Regs.AQCTLB.bit.CBD = AQ_SET;
183
184     // setup the Time-Base Period Register (TBPRD)
```

Microcontrôleurs

- µC : microcontroller = Microprocessor (µP) + memory + peripherals = PC in 1 chip
- DSP : Digital Signal Processor
  - Microprocessor (µP), heart of a computer
  - Additional hardware modules to accelerate some computation operations: Sum Of Product (SOP)

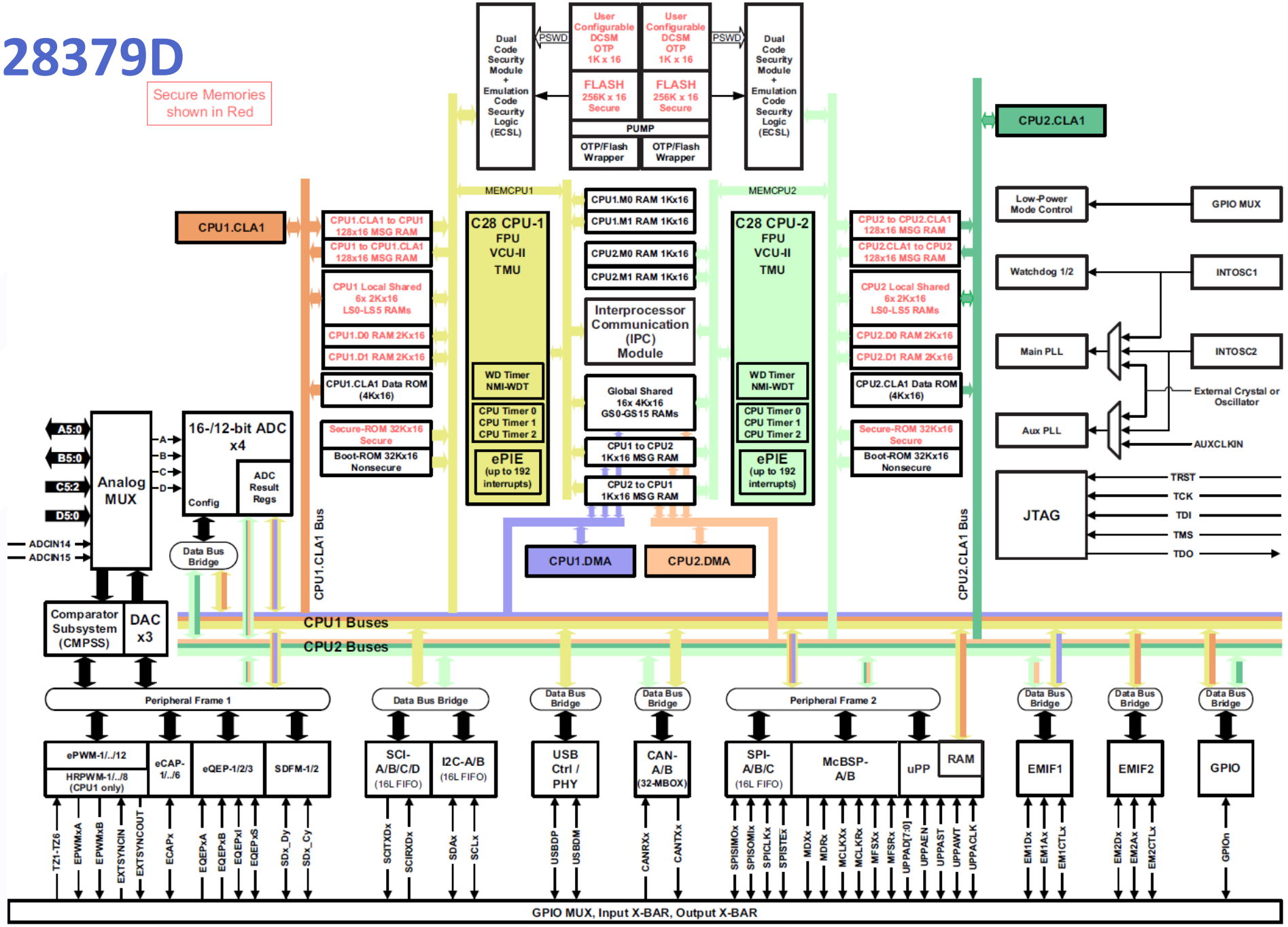
$$y = \sum_{i=0}^{3} coeff[i] * data[i]$$

- DSC : Digital Signal Controller
  - Microcontroller (µC) + DSP engine in 1 chip
  - Combine the power and computation of a DSP with the memory and peripherals of a µC

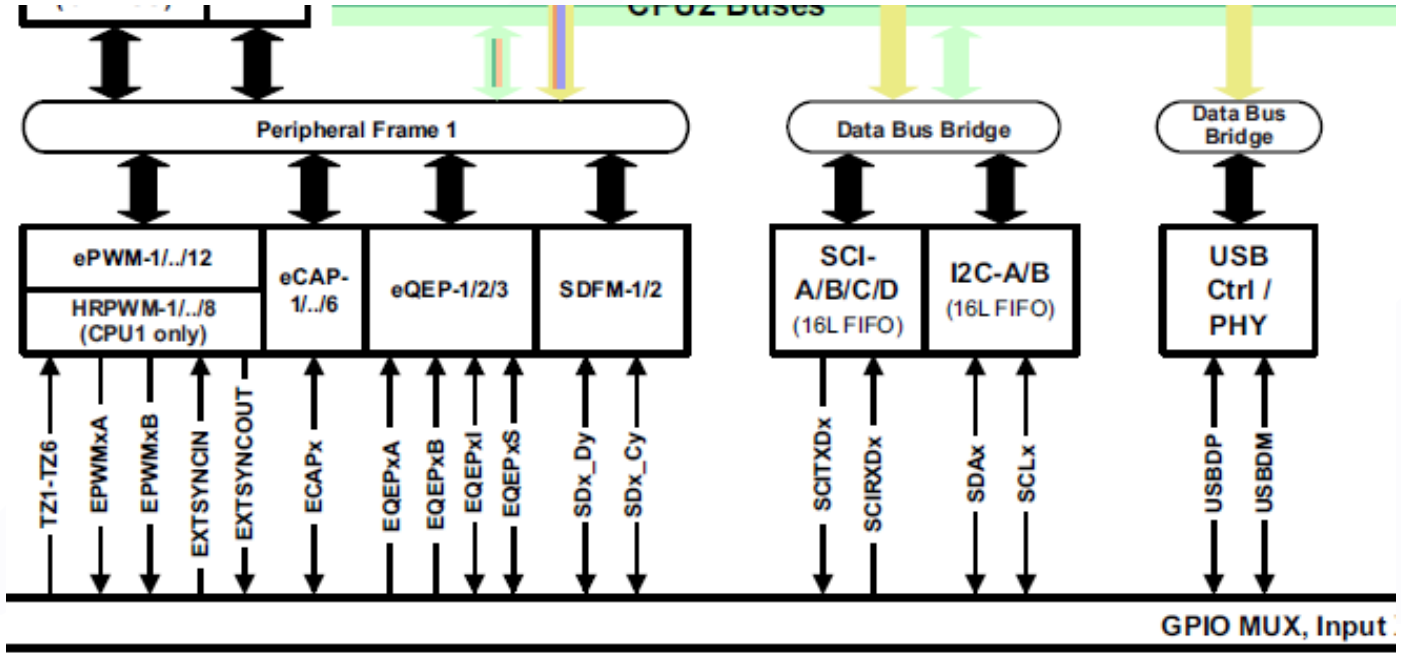
- The DSC is the optimal solution for real time control of embedded systems that requires high computation
- The µC is the optimal solution for Internet of Things systems that requires low cost, high connectivity and many interfaces

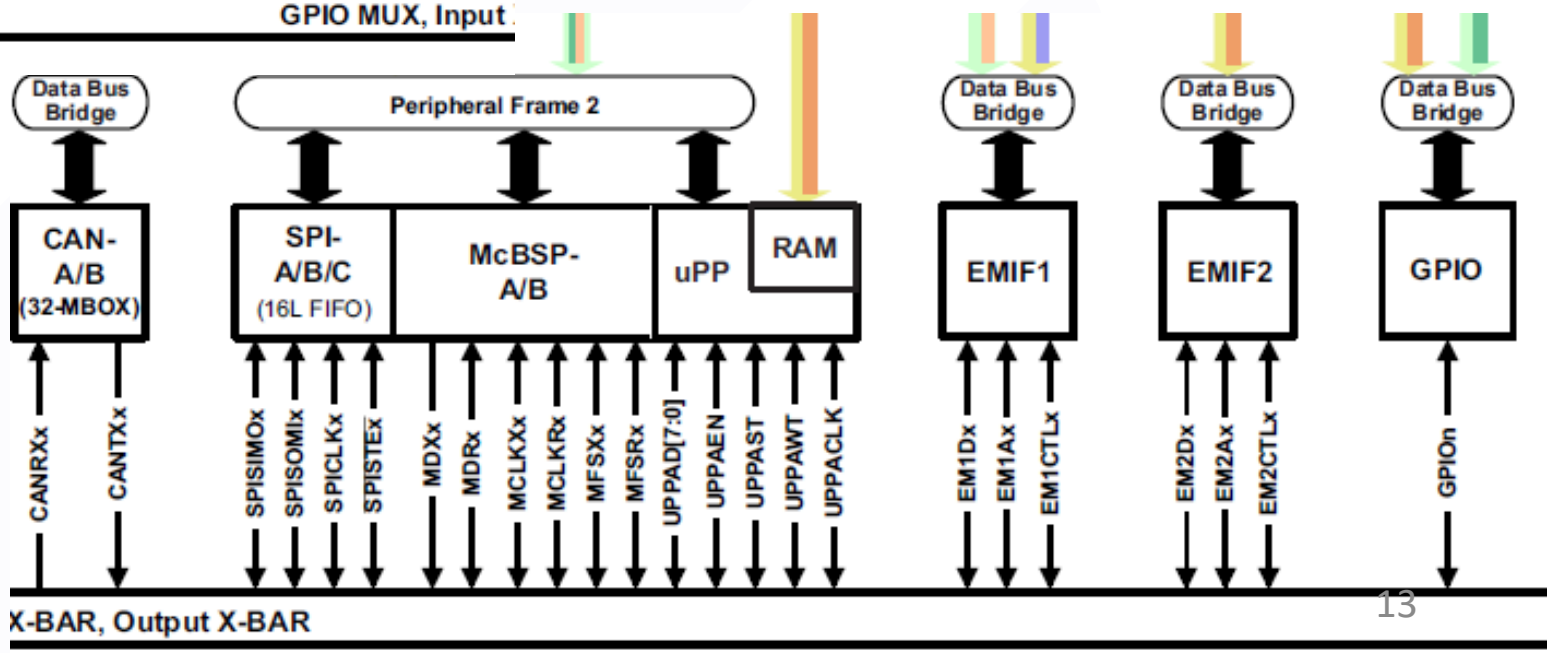
DSC F28379D

Secure Memories shown in Red



DSC F28379D


Peripheral Frame 1
Data Bus Bridge
Data Bus Bridge
ePWM-1/../12
HRPWM-1/../8
(CPU1 only)
eCAP-
1/../6
eQEP-1/2/3
SDFM-1/2
SCI-
A/B/C/D
(16L FIFO)
I2C-A/B
(16L FIFO)
USB
Ctrl /
PHY
TZ1-TZ6
EPWMxA
EPWMxB
EXTSYNCIN
EXTSYNCOUT
ECAPx
EQEPxA
EQEPxB
EQEPxI
EQEPxS
SDx_Dy
SDx_Cy
SCITXDx
SCIRXDx
SDAx
SCLx
USBDP
USBDM
GPIO MUX, Input


Data Bus
Bridge
Peripheral Frame 2
Data Bus
Bridge
Data Bus
Bridge
Data Bus
Bridge
CAN-
A/B
(32-MBOX)
SPI-
A/B/C
(16L FIFO)
McBSP-
A/B
uPP
RAM
EMIF1
EMIF2
GPIO
CANRXx
CANTXx
SPISIMOx
SPISOMIx
SPICLKx
SPISTEx
MDXx
MDRx
MCLKXx
MCLKRx
MFSXx
MFSRx
UPPAD[7:0]
UPPAEN
UPPAST
UPPAWT
UPPACLK
EM1Dx
EM1Ax
EM1CTLx
EM2Dx
EM2Ax
EM2CTLx
GPIOn
X-BAR, Output X-BAR

Features

- Dual-core architecture
  - Two TMS320C28x 32-bit CPUs, – 200 MHz
  - IEEE 754 single-precision Floating-Point Unit (FPU)
  - Trigonometric Math Unit (TMU)
  - Viterbi/Complex Math Unit (VCU-II)
- Two programmable Control Law Accelerators (CLAs)
  - 200 MHz – IEEE 754 FP instructions
  - Executes code independently of main CPU
- On-chip memory
  - 512KB (256KW) or 1MB (512KW) of flash (ECC-protected)
  - 172KB (86KW) or 204KB (102KW) of RAM
  - Dual-zone security supporting third-party development
  - Unique identification number
- Clock and system control
  - Two internal zero-pin 10-MHz oscillators
  - On-chip crystal oscillator
  - Windowed watchdog timer module
  - Missing clock detection circuitry
- 1.2-V core, 3.3-V I/O design
- System peripherals
  - Two External Memory Interfaces (EMIFs) with ASRAM and SDRAM support
  - Dual 6-channel Direct Memory Access (DMA) controllers
  - Up to 169 individually programmable, multiplexed General-Purpose Input/Output (GPIO) pins with input filtering
  - Expanded Peripheral Interrupt controller (ePIE)
  - Multiple Low-Power Mode (LPM) support with external wakeup
- Communications peripherals
  - USB 2.0 (MAC + PHY)
  - Support for 12-pin 3.3 V-compatible Universal Parallel Port (uPP) interface
  - Two Controller Area Network (CAN) modules
  - Three high-speed (up to 50-MHz) SPI ports (pin-bootable)
  - Two Multichannel Buffered Serial Ports (McBSPs)
  - Four Serial Communications Interfaces (SCI/UART) (pin-bootable)
  - Two I2C interfaces (pin-bootable)

Features

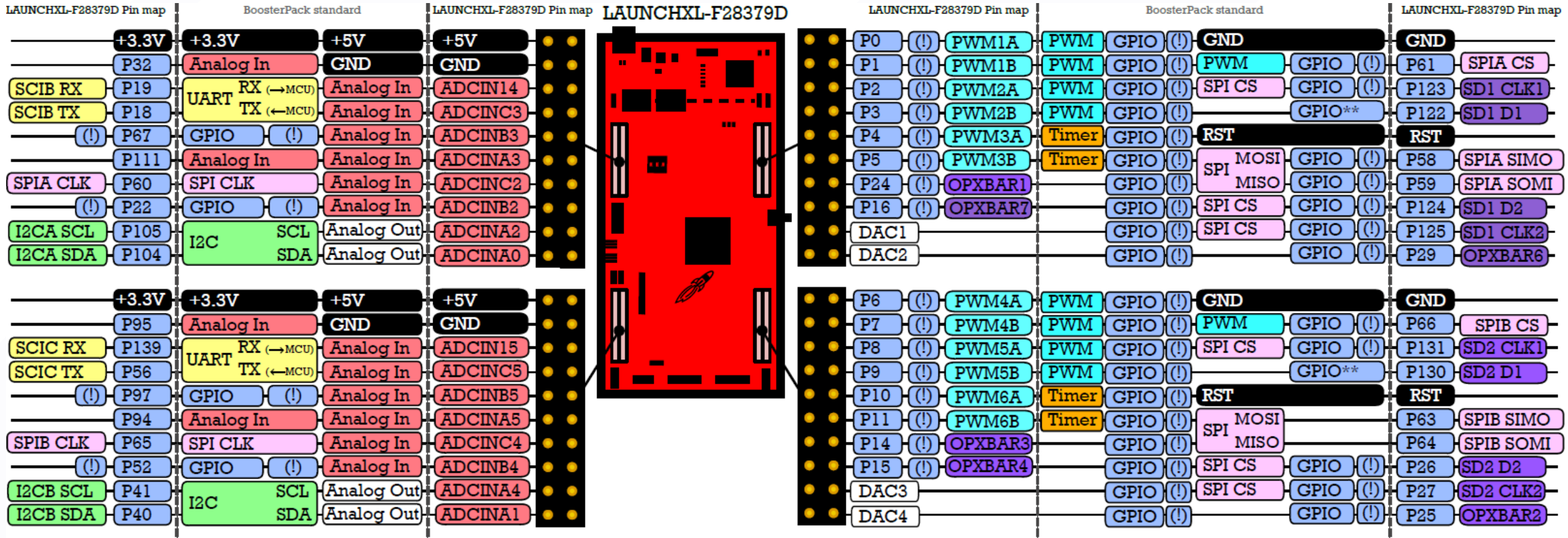
- Analog subsystem
  - – Up to four Analog-to-Digital Converters (ADCs)
    - 16-bit mode
      - – 1.1 MSPS each (up to 4.4-MSPS system throughput)
      - – Differential inputs
      - – Up to 12 external channels
    - 12-bit mode
      - – 3.5 MSPS each (up to 14-MSPS system throughput)
      - – Single-ended inputs
      - – Up to 24 external channels
    - Single Sample-and-Hold (S/H) on each ADC
    - Hardware-integrated post-processing of ADC conversions
      - – Saturating offset calibration
      - – Error from setpoint calculation
      - – High, low, and zero-crossing compare, with interrupt capability
      - – Trigger-to-sample delay capture
  - – Eight windowed comparators with 12-bit Digitalto-Analog Converter (DAC) references
  - – Three 12-bit buffered DAC outputs
- Enhanced control peripherals
  - – 24 Pulse Width Modulator (PWM) channels with enhanced features
  - – 16 High-Resolution Pulse Width Modulator (HRPWM) channels
    - High resolution on both A and B channels of 8 PWM modules
    - Dead-band support (on both standard and high resolution)
  - – Six Enhanced Capture (eCAP) modules
  - – Three Enhanced Quadrature Encoder Pulse (eQEP) modules
  - – Eight Sigma-Delta Filter Module (SDFM) input channels, 2 parallel filters per channel
    - Standard SDFM data filtering

- PWM / ADC / GPIO

Questions à propos des périphériques PWM / ADC / GPIO / SPI / I2C

- What : Qu'est ce que c'est ?
- How : Comment fonctionne t il ?
- Usage : Comment l'utilise t on ?
- Target : Les applications possibles

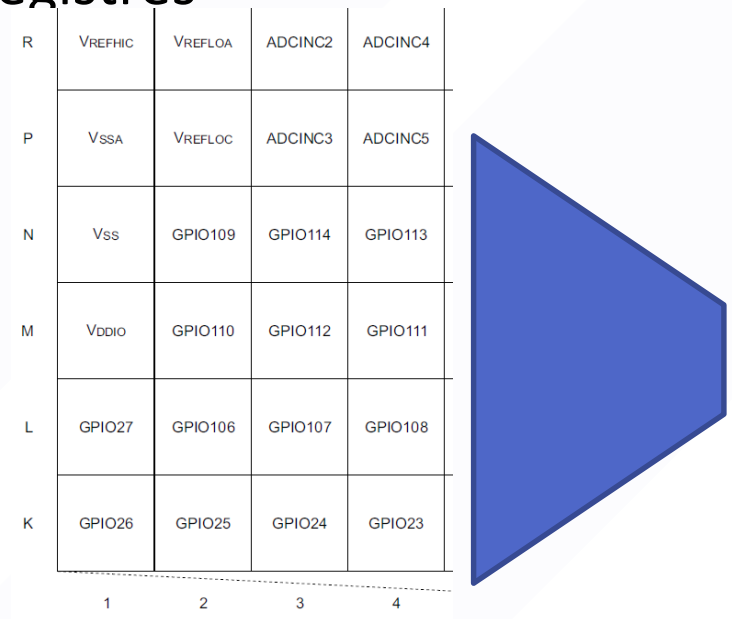
J1.01 +3.3V	J3.21 +5V	J4.40 PWM1A PWMHA	J2.20 GND
J1.02 P32 Info	J3.20 GND	J4.39 PWM1B PWMLA	J2.19 P61
J1.03 P19 FAULT (in)	J3.23 ADCIN14 Va	J4.38 PWM2A PWMHB	J2.18 P123 SCS
J1.04 P18 HallA	J3.24 ADCINC3 Vb	J4.37 PWM2B PWMLB	J2.17 P122
J1.05 P67 HallB	J3.25 ADCINB3 Vc	J4.36 PWM3A PWMHC	J2.16 RESET
J1.06 P111 HallC	J3.26 ADCINA3 Vdc	J4.35 PWM3B PWMLC	J2.15 P58 MOSI
J1.07 P60 SCLK	J3.27 ADCINC2 Ia	J4.34 P24	J2.14 P59 MISO
J1.08 P22	J3.28 ADCINB2 Ib	J4.33 P16	J2.13 P124 ENGATE
J1.09 P105 SCL_OLED	J3.29 ADCINA2 Ic	J4.32 DAC1 BNC1	J2.12 P125 WAKE
J1.10 P104 SDA_OLED	J3.30 ADCINA0	J4.31 DAC2 BNC2	J2.11 P29

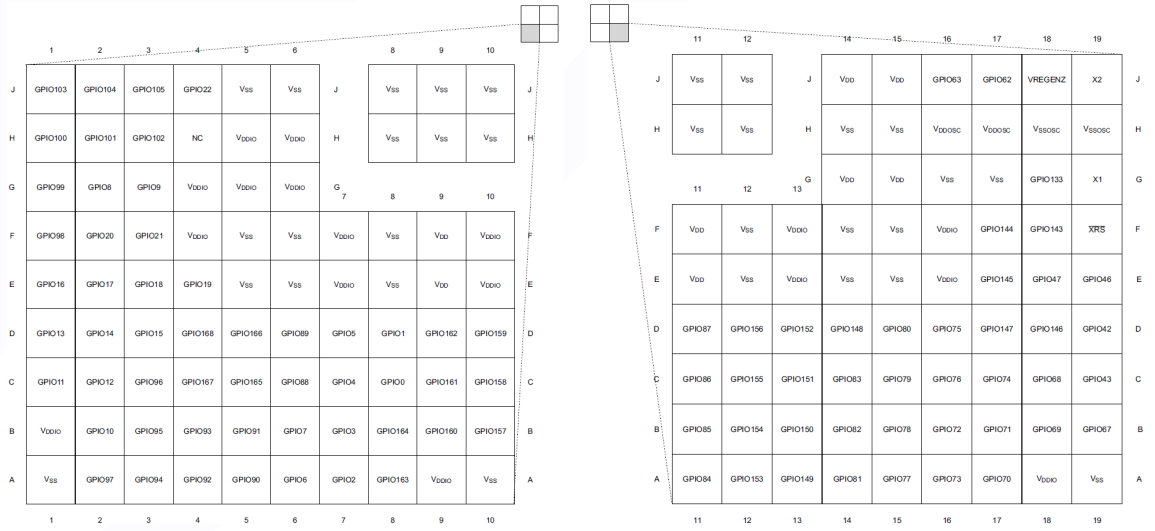


GPIO : General-Purpose Digital I/O

What : Qu'est ce que c'est ?

- Entrées ou sorties logiques
- Signal logique 1/0 ⇔ Signal de tension 3.3V / 0V
- Pins (broches) multiplexées avec d'autres périphériques
- Contrôlées par des registres
- Pullup interne
- Interruptions

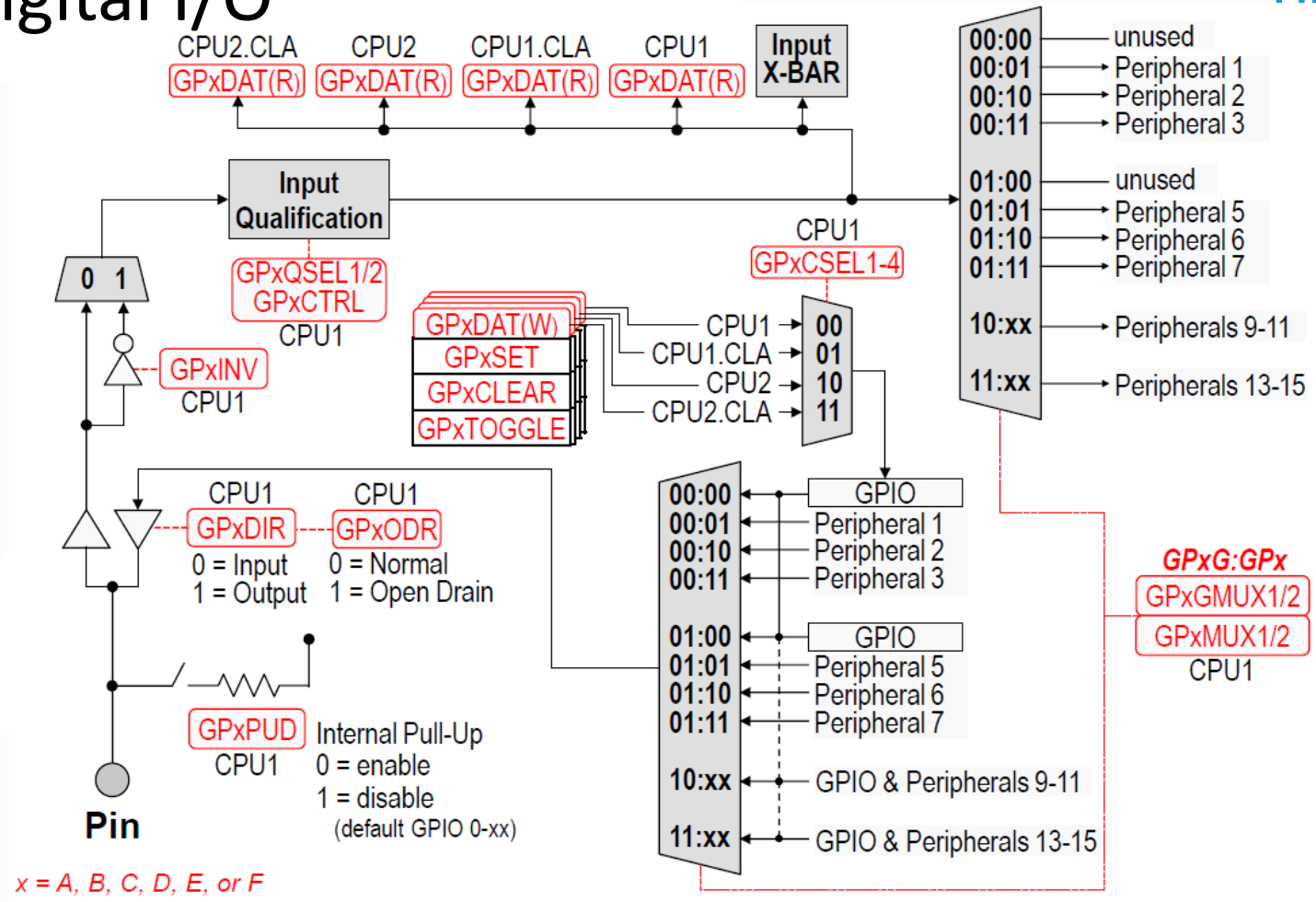




GPIO : General-Purpose Digital I/O

How :
Comment
fonctionne t il ?

A GPIO Group multiplexer and four GPIO Index multiplexers provide a double layer of multiplexing to allow up to twelve independent peripheral signals and a digital I/O function to share a single pin. Each output pin can be controlled by either a peripheral or CPU1, CPU1 CLA, CPU2, or CPU2 CLA. However, the peripheral multiplexing and pin assignment can only be configured by CPU1.



If the pin is set as a GPIO by the GPIO multiplexer, the direction will be set by the GPIO direction register. The GPIO data register will have the value of the pin if set as an input or write the value of the data register to the pin if set as an output. The data register can be quickly and easily modified using set, clear, or toggle registers. Also, the pin has an option for an internal pull-up.

GPIO : General-Purpose Digital I/O

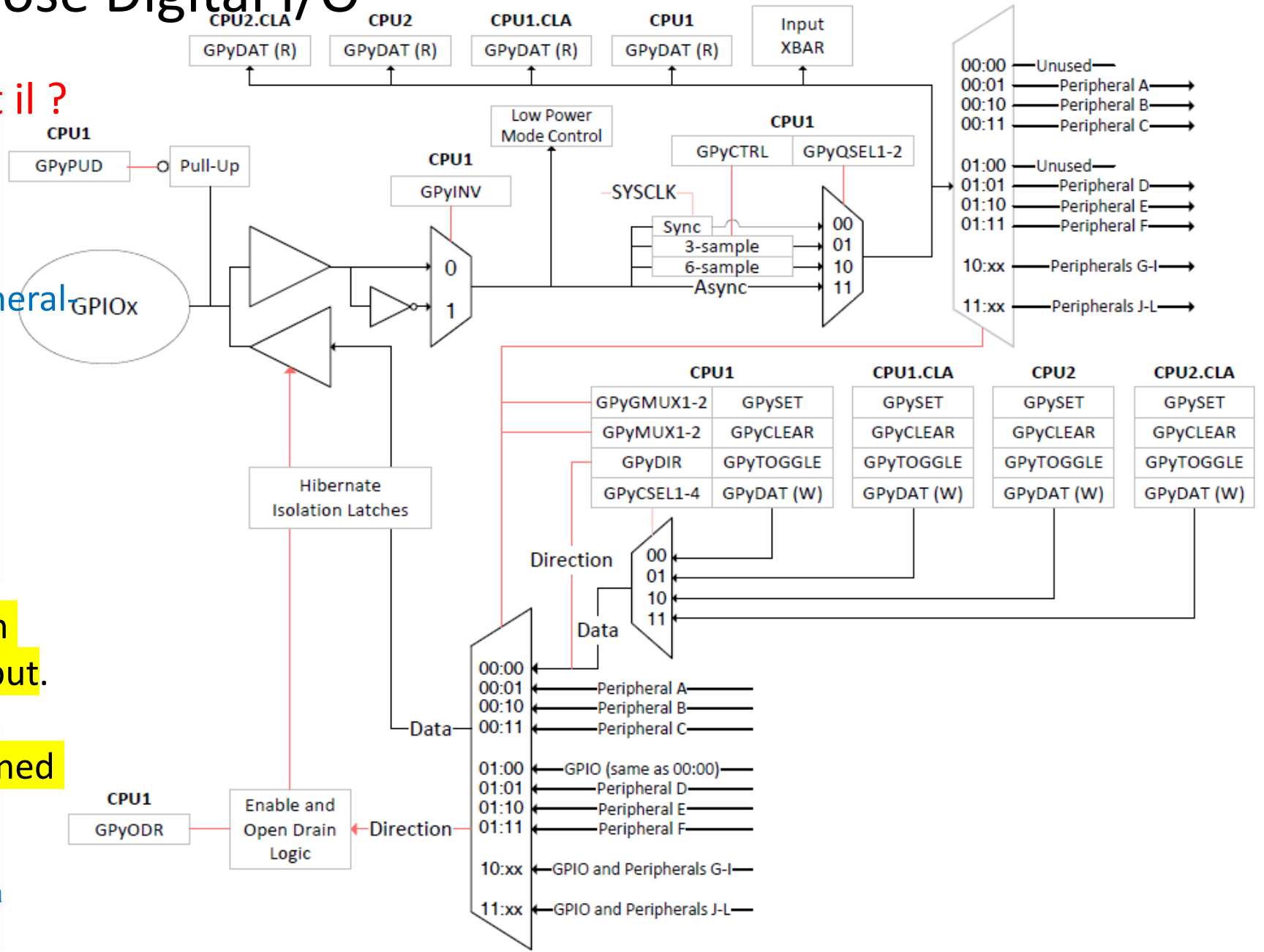
How : Comment fonctionne t il ? (Détails)

Each general-purpose I/O pin has a maximum of four options, either general-purpose I/O or up to three possible peripheral pin assignments.

This is selected using the GPIO port multiplexer.

If the pin is set to GPIO, the direction register sets it as an input or an output.

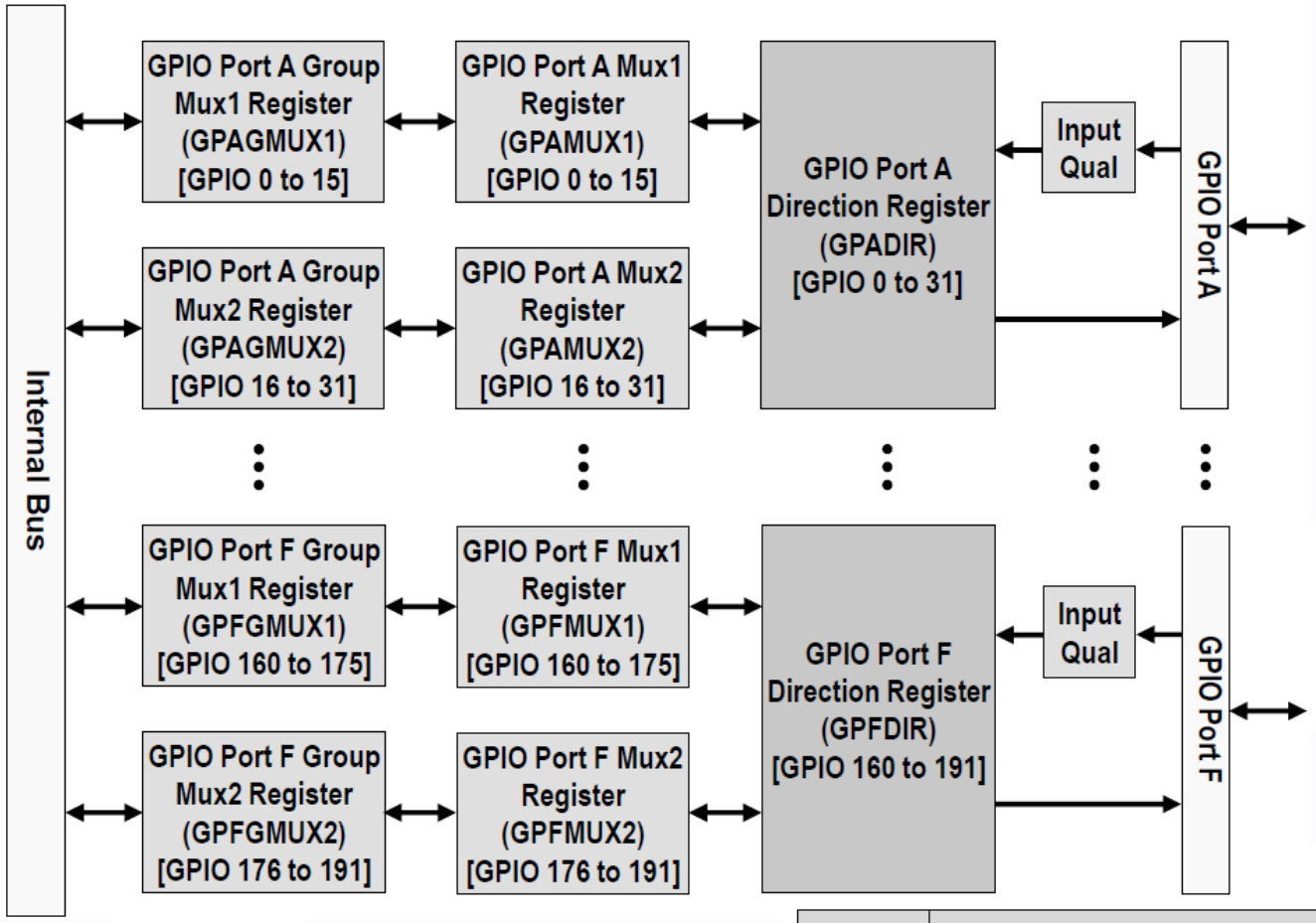
The input qualification will be explained shortly.



TMS320F2837xD Technical Reference Manua
TMS320F2837xD Microcontroller Workshop

GPIO : General-Purpose Digital I/O

How :
Comment fonctionne t il ?



	GPIO Mux Selection							
GPIO Index	0, 4, 8, 12	1	2	3	5	6	7	15
GPyGMUXn. GPIOz =	00b, 01b, 10b, 11b	00b			01b			11b
GPyMUXn. GPIOz =	00b	01b	10b	11b	01b	10b	11b	11b
	GPIO0	EPWM1A (O)				SDAA (I/OD)		
	GPIO1	EPWM1B (O)		MFSRB (I/O)		SCLA (I/OD)		
	GPIO2	EPWM2A (O)			OUTPUTXBAR1 (O)	SDAB (I/OD)		
	GPIO3	EPWM2B (O)	OUTPUTXBAR2 (O)	MCLKRB (I/O)	OUTPUTXBAR2 (O)	SCLB (I/OD)		
	GPIO4	EPWM3A (O)			OUTPUTXBAR3 (O)	CANTXA (O)		

Groupés en 6 ports : GPIO Port A. .GPIO Port F
Configurables via GPAMUX1,…, GPADIR,… GPAPUD

TMS320F2837xD Technical Reference Manual spruhm8i
TMS320F2837xD Microcontroller Workshop

GPIO : General-Purpose Digital I/O

How :
Comment
fonctionne t il ?

GPAGMUX1[13:12]	GPAMUX1[13:12]	Pin functionality
00	00	GPIO6
00	01	EPWM4A
00	10	OUTPUTXBAR4
00	11	EXTSYNCOUT
01	00	GPIO6
01	01	EQEP3A
01	10	CANB_TX
01	11	
10	00	GPIO6
10	01	
10	10	
10	11	
11	00	GPIO6
11	01	
11	10	
11	11	

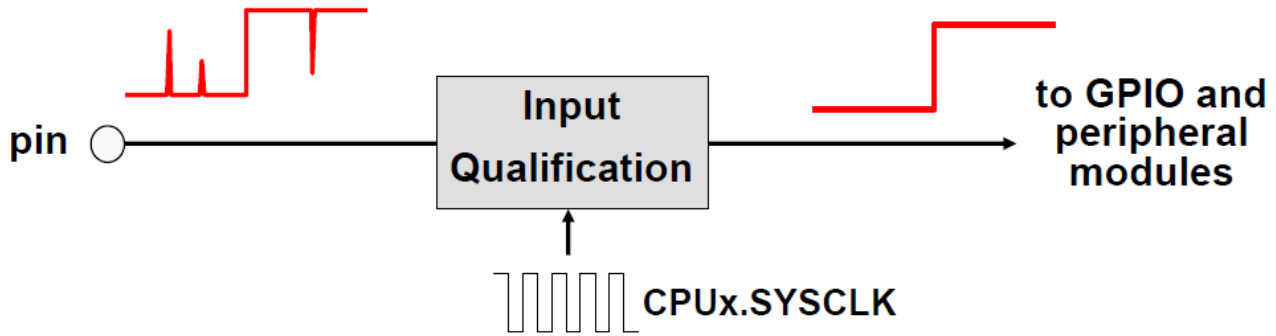
	GPIO Mux Selection							
GPIO Index	0, 4, 8, 12	1	2	3	5	6	7	15
GPyGMUXn. GPIOz =	00b, 01b, 10b, 11b	00b				01b		11b
GPyMUXn. GPIOz =	00b	01b	10b	11b	01b	10b	11b	11b
	GPIO0	EPWM1A (O)				SDAA (I/OD)		
	GPIO1	EPWM1B (O)		MFSRB (I/O)		SCLA (I/OD)		
	GPIO2	EPWM2A (O)			OUTPUTXBAR1 (O)	SDAB (I/OD)		
	GPIO3	EPWM2B (O)	OUTPUTXBAR2 (O)	MCLKRB (I/O)	OUTPUTXBAR2 (O)	SCLB (I/OD)		
	GPIO4	EPWM3A (O)			OUTPUTXBAR3 (O)	CANTXA (O)		
	GPIO5	EPWM3B (O)	MFSRA (I/O)	OUTPUTXBAR3 (O)		CANRXA (I)		
	GPIO6	EPWM4A (O)	OUTPUTXBAR4 (O)	EXTSYNCOUT (O)	EQEP3A (I)	CANTXB (O)		

For example, the multiplexing for the GPIO 6 pin is controlled by writing to GPAGMUX[13:12] and GPAMUX[13:12]
By writing to these bits, GPIO 6 can be configured as either a general-purpose digital I/O or one of four different peripheral functions

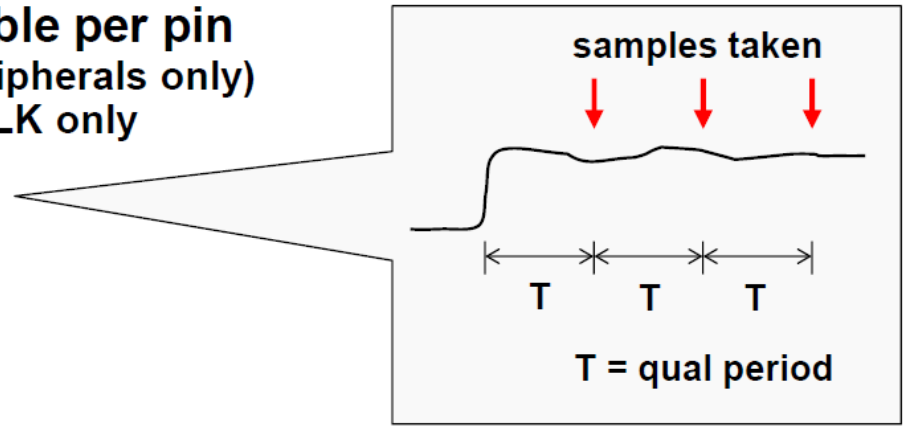
TMS320F2837xD Technical Reference Manual spruhm8i

GPIO : General-Purpose Digital I/O

How :
Comment
fonctionne t il ?



- Qualification available on ports A - F
- Individually selectable per pin
  - no qualification (peripherals only)
  - sync to CPUx.SYSCLK only
  - qualify 3 samples
  - qualify 6 samples



The GPIO input qualification feature allows filtering out noise on a pin. The user would select the number of samples and qualification period.
Qualification is available on ports A - B and is individually selectable per pin

GPIO : General-Purpose Digital I/O

Usage : Comment l'utilise t on ?

Step 1. **Plan the device pin-out:**

Choose peripheral mux with GPyMUX1/2, GPyGMUX1/2 registers

Step 2. **Enable or disable internal pull-up resistors:**

(GPyPUD) registers.

Step 3. **Select input qualification:** (Optional)

Input only, GPyCTRL, GPyQSEL1, GPyQSEL2 registers

Step 4. **For digital general purpose I/O, select the direction of the pin:**

GPyDIR registers.

Use then GPyCLEAR, GPySET, or GPyTOGGLE registers

Acronym	Register Name
GPACTRL	GPIO A Qualification Sampling Period Control (GPIO0 to 31)
GPAQSEL1	GPIO A Qualifier Select 1 Register (GPIO0 to 15)
GPAQSEL2	GPIO A Qualifier Select 2 Register (GPIO16 to 31)
GPAMUX1	GPIO A Mux 1 Register (GPIO0 to 15)
GPAMUX2	GPIO A Mux 2 Register (GPIO16 to 31)
GPADIR	GPIO A Direction Register (GPIO0 to 31)
GPAPUD	GPIO A Pull Up Disable Register (GPIO0 to 31)
GPAINV	GPIO A Input Polarity Invert Registers (GPIO0 to 31)
GPAODR	GPIO A Open Drain Output Register (GPIO0 to GPIO31)
GPAGMUX1	GPIO A Peripheral Group Mux (GPIO0 to 15)
GPAGMUX2	GPIO A Peripheral Group Mux (GPIO16 to 31)
GPACSEL1	GPIO A Core Select Register (GPIO0 to 7)
GPACSEL2	GPIO A Core Select Register (GPIO8 to 15)
GPACSEL3	GPIO A Core Select Register (GPIO16 to 23)
GPACSEL4	GPIO A Core Select Register (GPIO24 to 31)
GPALOCK	GPIO A Lock Configuration Register (GPIO0 to 31)
GPACR	GPIO A Lock Commit Register (GPIO0 to 31)
GPBCTRL	GPIO B Qualification Sampling Period Control (GPIO32 to 63)
GPBQSEL1	GPIO B Qualifier Select 1 Register (GPIO32 to 47)
GPBQSEL2	GPIO B Qualifier Select 2 Register (GPIO48 to 63)
GPBMUX1	GPIO B Mux 1 Register (GPIO32 to 47)
GPBMUX2	GPIO B Mux 2 Register (GPIO48 to 63)
GPBDIR	GPIO B Direction Register (GPIO32 to 63)
GPBPUD	GPIO B Pull Up Disable Register (GPIO32 to 63)

Optional :

Step 5. **Select low power mode wake-up sources:**

Specify which pins (0-63), that will be able to wake the device from HALT and STANDBY. GPIOLPMSEL0/1 registers.

Step 6. **Select external interrupt sources:**

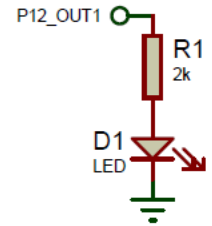
Specify the source for the XINT1 - 5 interrupts. GPIOXINTnSEL, XINTnCR registers.

TMS320F2837xD Technical Reference Manual spruhm8i p.953, p.966

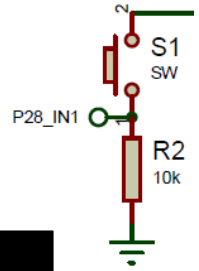
GPIO : General-Purpose Digital I/O

Usage : Comment l'utilise t on ?

```
EALLOW;
GpioCtrlRegs.GPAPUD.bit.GPIO0 = 1;    // Disable pull-up on GPIO0 (EPWM1A)
GpioCtrlRegs.GPAPUD.bit.GPIO1 = 1;    // Disable pull-up on GPIO1 (EPWM1B)
GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;   // Configure GPIO0 as EPWM1A
GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;   // Configure GPIO1 as EPWM1B

GpioCtrlRegs.GPAPUD.bit.GPIO2 = 1;    // Disable pull-up on GPIO2 (EPWM2A)
GpioCtrlRegs.GPAPUD.bit.GPIO3 = 1;    // Disable pull-up on GPIO3 (EPWM2B)
GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // Configure GPIO2 as EPWM2A
GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;   // Configure GPIO3 as EPWM2B

GpioCtrlRegs.GPAPUD.bit.GPIO4 = 1;    // Disable pull-up on GPIO4 (
GpioCtrlRegs.GPAPUD.bit.GPIO5 = 1;    // Disable pull-up on GPIO5 (
GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;   // Configure GPIO4 as EPWM3A
GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;   // Configure GPIO5 as EPWM3B

GpioCtrlRegs.GPBPUD.bit.GPIO32 = 1;     // Disable pull-up on GPIO32
GpioCtrlRegs.GPBMUX1.bit.GPIO32 = 0;    // Configure GPIO32 as GPIO
GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1;     // GPIO32 = output
GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;   // Clear
EDIS;
```




```
// Set
GpioDataRegs.GPASET.bit.GPIO32 = 1;
// Clear
GpioDataRegs.GPACLEAR.bit.GPIO32 = 1;
// Toggle
GpioDataRegs.GPATOGGLE.bit.GPIO32 = 1;
```

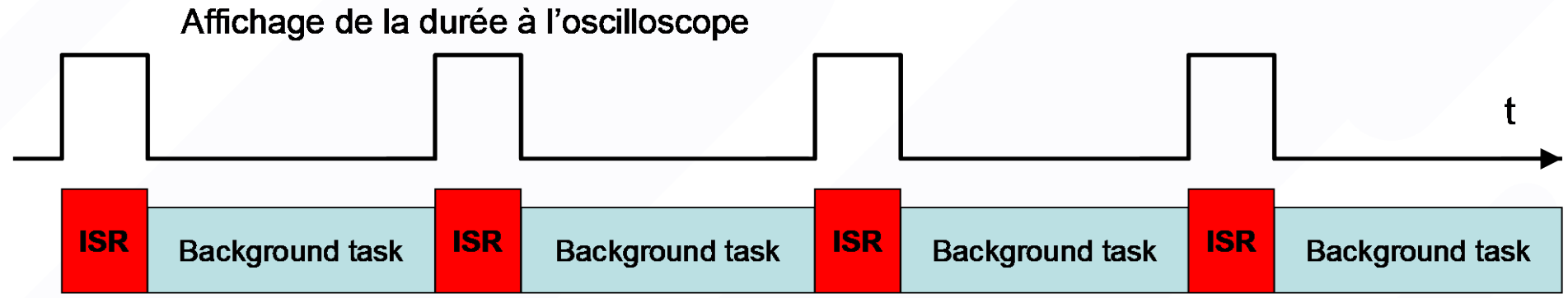



```
// Read
int val = GpioDataRegs.GPADAT.bit.GPIO28;
```

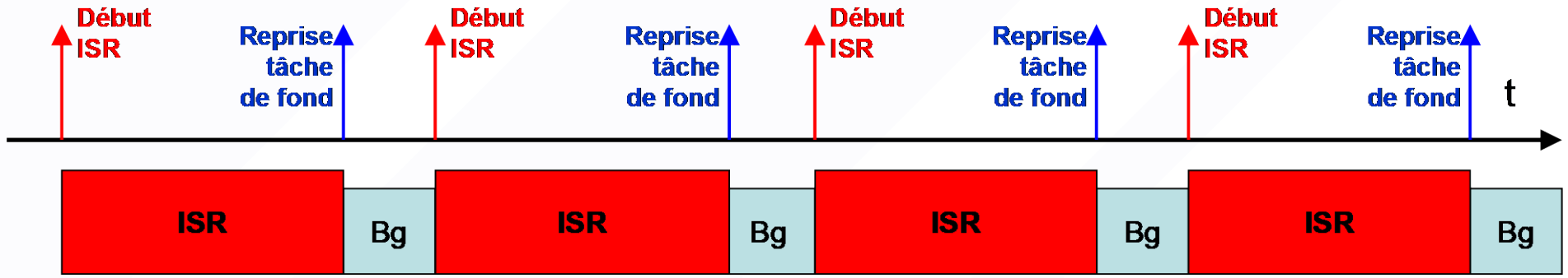
Interruptions

What : Qu'est ce que c'est ?

- Une interruption est une suspension temporaire de l'exécution d'un programme informatique par le microprocesseur afin d'exécuter un programme prioritaire (ISR)
- ISR : Interrupt Service Routine ou Routine de Service de l'Interruption



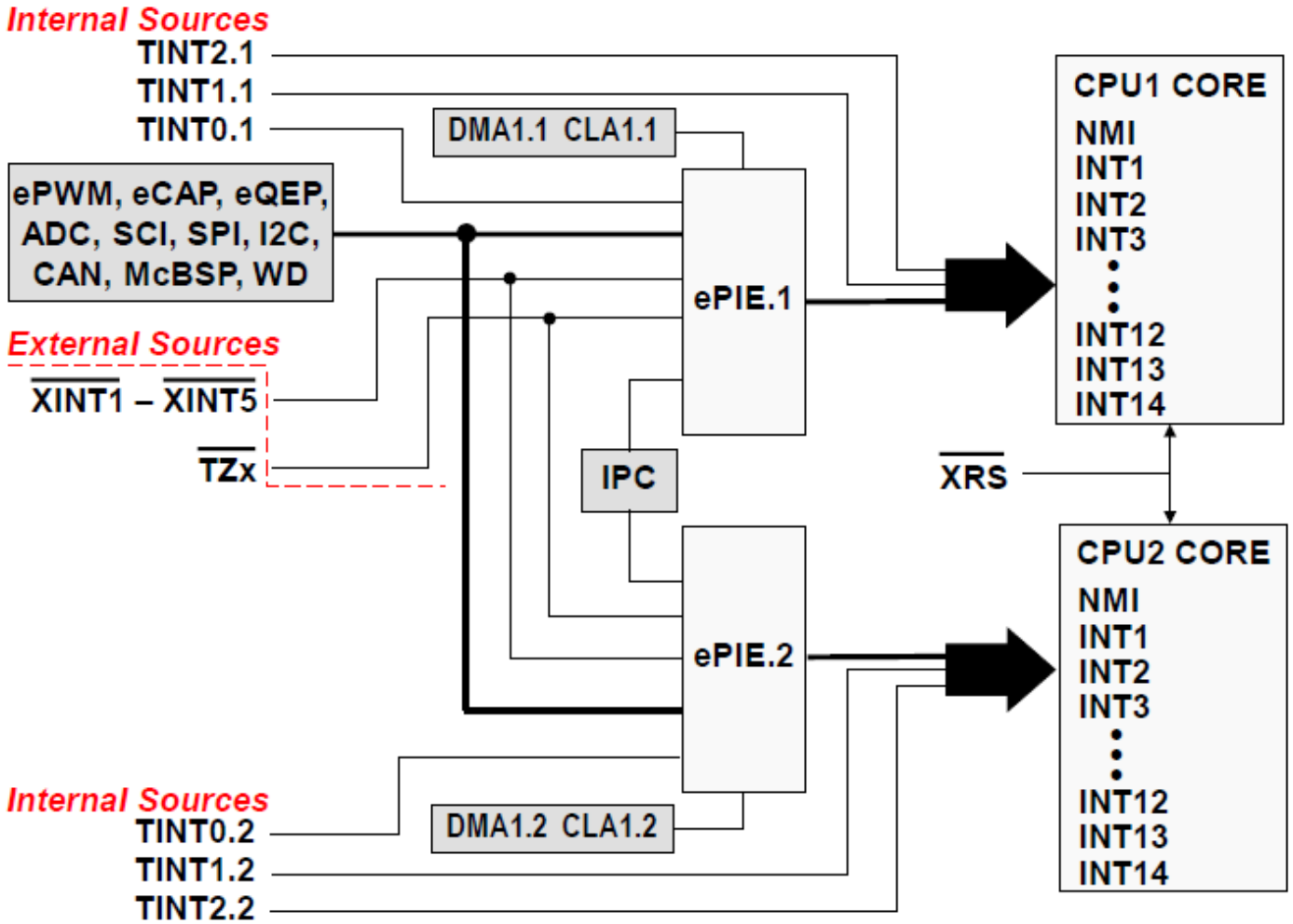
Tâche peu gourmande en temps CPU



Interruptions

How : Comment fonctionnent elles ?

Each C28x CPU core in the F2837xD device has its own PIE module, and each PIE module is configured independently.
Some interrupt signals are sourced from shared peripherals that can be owned by either CPU, and these interrupt signals are sent to both CPU PIE modules regardless of which CPU owns the peripheral.
Therefore, if enabled a peripheral owned by one CPU can cause an interrupt on the other CPU.



Interruptions

How : Comment fonctionnent elles ?

It is easier to explain the interrupt processing flow from the core back out to the interrupt sources.

The INTM is the master (global) interrupt switch.
This switch must be closed for any interrupts to propagate into the core.

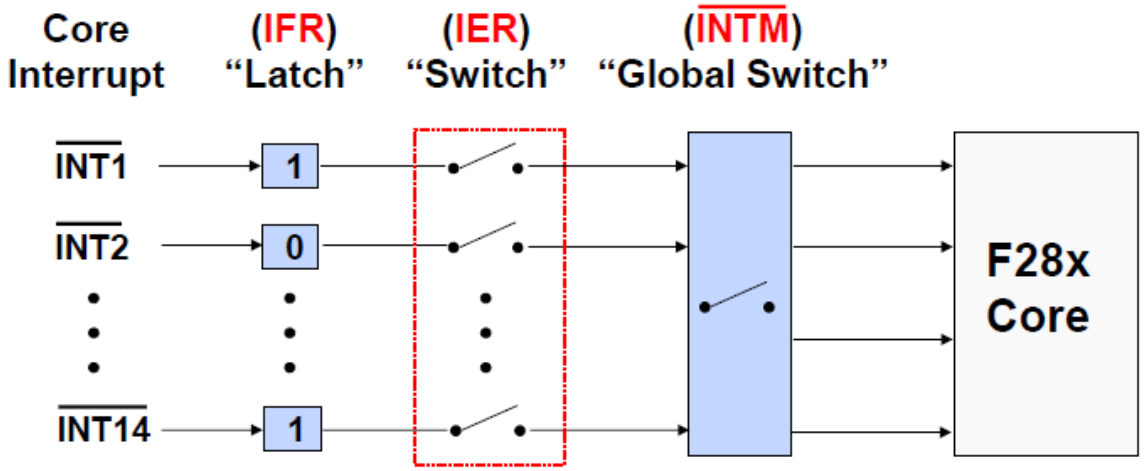
The next layer out is the interrupt enable register (IER).
The appropriate interrupt line switch must be closed to allow an interrupt through.

The interrupt flag register (IFR) gets set when an interrupt occurs.

Once the core starts processing an interrupt, the INTM switch opens to avoid nested interrupts and the flag is cleared.

Maskable Interrupt Processing

Conceptual Core Overview



- A valid signal on a specific interrupt line causes the latch to display a "1" in the appropriate bit
- If the individual and global switches are turned "on" the interrupt reaches the core

Interruptions

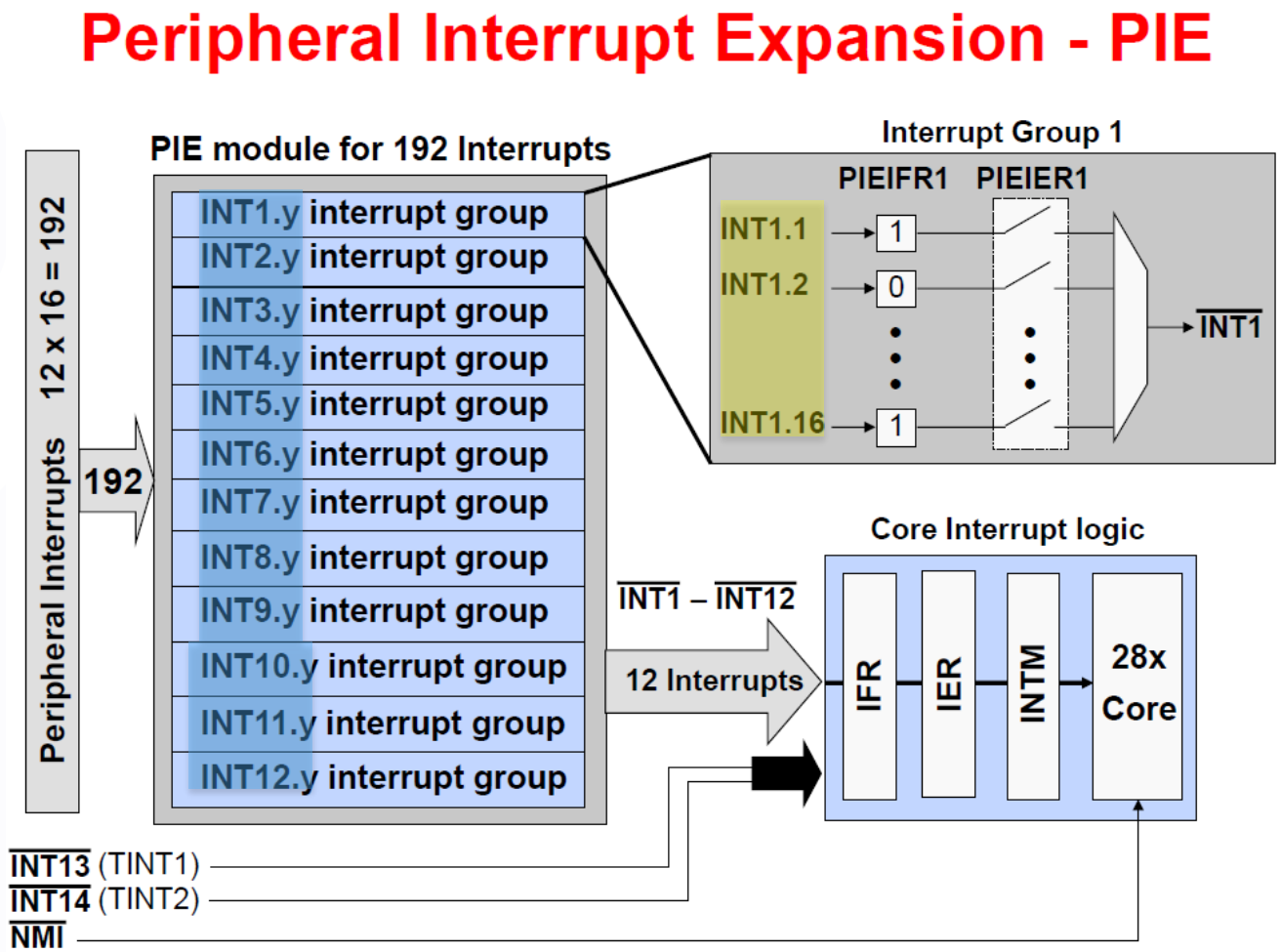
How : Comment fonctionnent elles ?

Now we need to look at the peripheral interrupt expansion block.
This block is connected to the core interrupt lines 1 through 12.
The PIE block consists of 12 groups.

Within each group, there are eight interrupt sources. Each group has a PIE interrupt enable register and a PIE interrupt flag register.

As you can see, the interrupts are numbered from 1.1 through 12.16, giving us a maximum of 192 interrupt sources.

Interrupt lines 13, 14, and NMI bypass the PIE block.



Interruptions

How : Comment fonctionnent elles ?

The interrupt assignment table tells us the location for each interrupt source within the PIE block. Notice the table is numbered from 1.1 through 12.16, perfectly matching the PIE block.

PIE Registers

PIEIFRx register (x = 1 to 12)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTx.16	INTx.15	INTx.14	INTx.13	INTx.12	INTx.11	INTx.10	INTx.9	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1

PIEIERx register (x = 1 to 12)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
INTx.16	INTx.15	INTx.14	INTx.13	INTx.12	INTx.11	INTx.10	INTx.9	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1

PIE Interrupt Acknowledge Register (PIEACK)

15 - 12	11	10	9	8	7	6	5	4	3	2	1	0
reserved	PIEACKx											

PIECTRL register

15 - 1	0
PIEVECT	ENPIE

F2837xD PIE Assignment Table - Lower

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKE	TINT0	ADCD1	XINT2	XINT1	*ADCC1*	ADCB1	ADCA1
INT2	PWM8_TZ	PWM7_TZ	PWM6_TZ	PWM5_TZ	PWM4_TZ	PWM3_TZ	PWM2_TZ	PWM1_TZ
INT3	PWM8	PWM7	PWM6	PWM5	PWM4	PWM3	PWM2	PWM1
INT4			ECAP6	ECAP5	ECAP4	ECAP3	ECAP2	ECAP1
INT5						EQEP3	EQEP2	EQEP1
INT6	MCBSP B_TX	MCBSP B_RX	MCBSP A_TX	MCBSP A_RX	SPIB_TX	SPIB_RX	SPIA_TX	SPIA_RX
INT7			DMA_CH6	DMA_CH5	DMA_CH4	DMA_CH3	DMA_CH2	DMA_CH1
INT8	SCID_TX	SCID_RX	SCIC_TX	SCIC_RX	I2CB_FIFO	I2CB	I2CA_FIFO	I2CA
INT9	CAN						A_TX	SCIA_RX
INT10	ADC						CA2	ADCA_EVT
INT11	CLA						A1_2	CLA1_1
INT12	FPU_						NT4	XINT3

F2837xD PIE Assignment Table - Upper

	INTx.16	INTx.15	INTx.14	INTx.13	INTx.12	INTx.11	INTx.10	INTx.9
INT1	*IPC3*	*IPC2*	*IPC1*	*IPC0*				
INT2					PWM12_TZ	PWM11_TZ	PWM10_TZ	PWM9_TZ
INT3					EPWM12	EPWM11	EPWM10	EPWM9
INT4								
INT5							SD2	SD1
INT6							SPIC_TX	SPIC_RX
INT7								
INT8		UPPA						
INT9		USBA						
INT10	ADCD4	ADCD3	ADCD2	ADCD_EVT	*ADCC4*	*ADCC3*	*ADCC2*	*ADCC_EVT*
INT11								
INT12	CLA_UF	CLA_OF	AUX_PLL_SLIP	SYS_PLL_SLIP	RAM_ACC_VIOLAT	FLASH_C_ERROR	RAM_C_ERROR	EMIF_ERROR

Interruptions

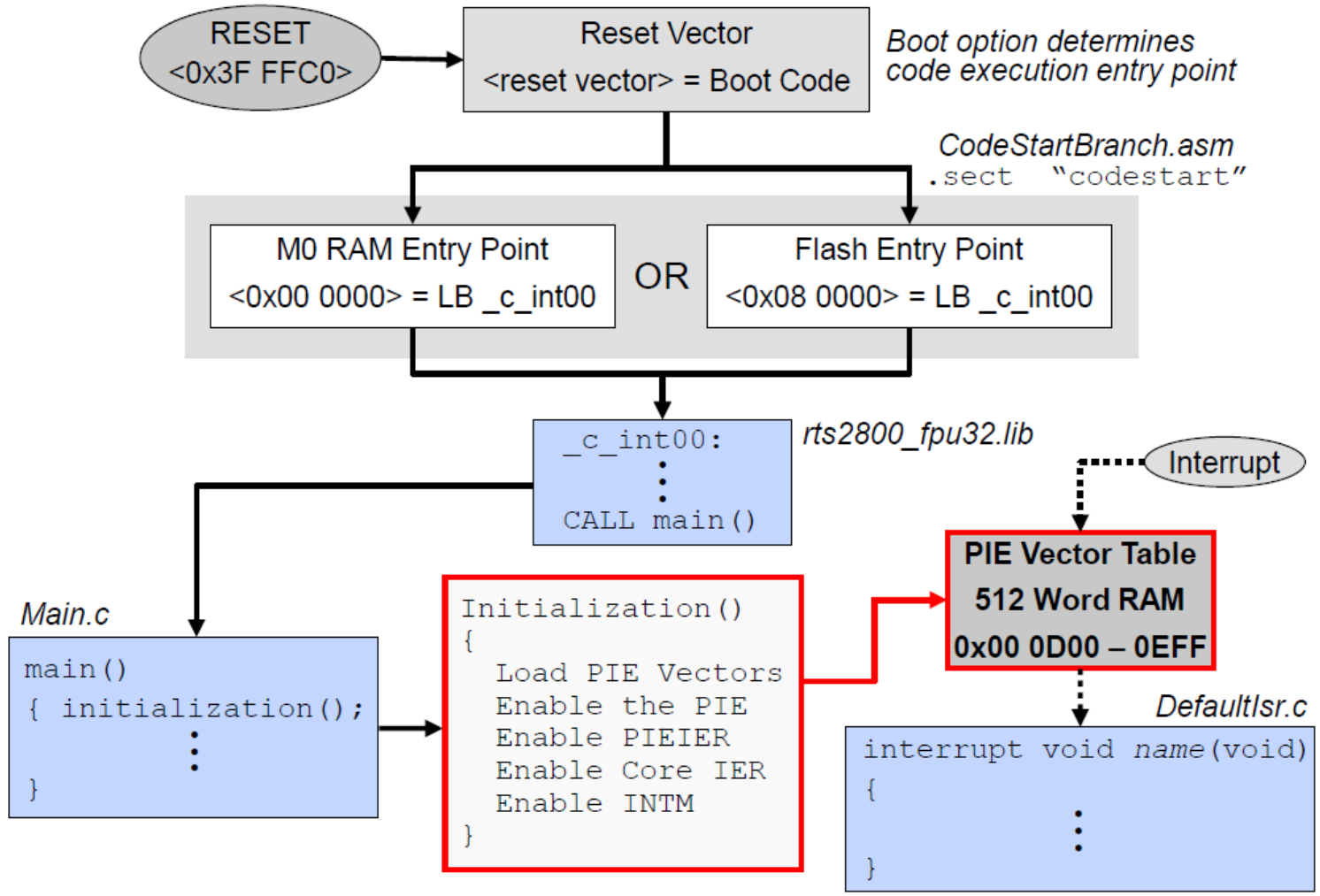
How : Comment fonctionnent elles ?

PIE Initialization Code Flow – Summary

After the device is reset and executes the boot coc
the selected boot option determines the code entı
point.
This figure shows two different entry points. The o
on the left is for memory block M0, and the one oı
right is for flash.

In either case, CodeStartBranch.asm has a “Long Branch” to the entry point of the runtime support library. After the runtime support library complete execution, it calls main.

In main, we have a function call to initialize the interrupt process and enable the PIE module.
When the CPU receives an interrupt, the vector address of the ISR is fetched from the PIE RAM, an the interrupt with the highest priority that is both flagged and enabled is executed.



Interruptions

How : Comment fonctionnent elles ?

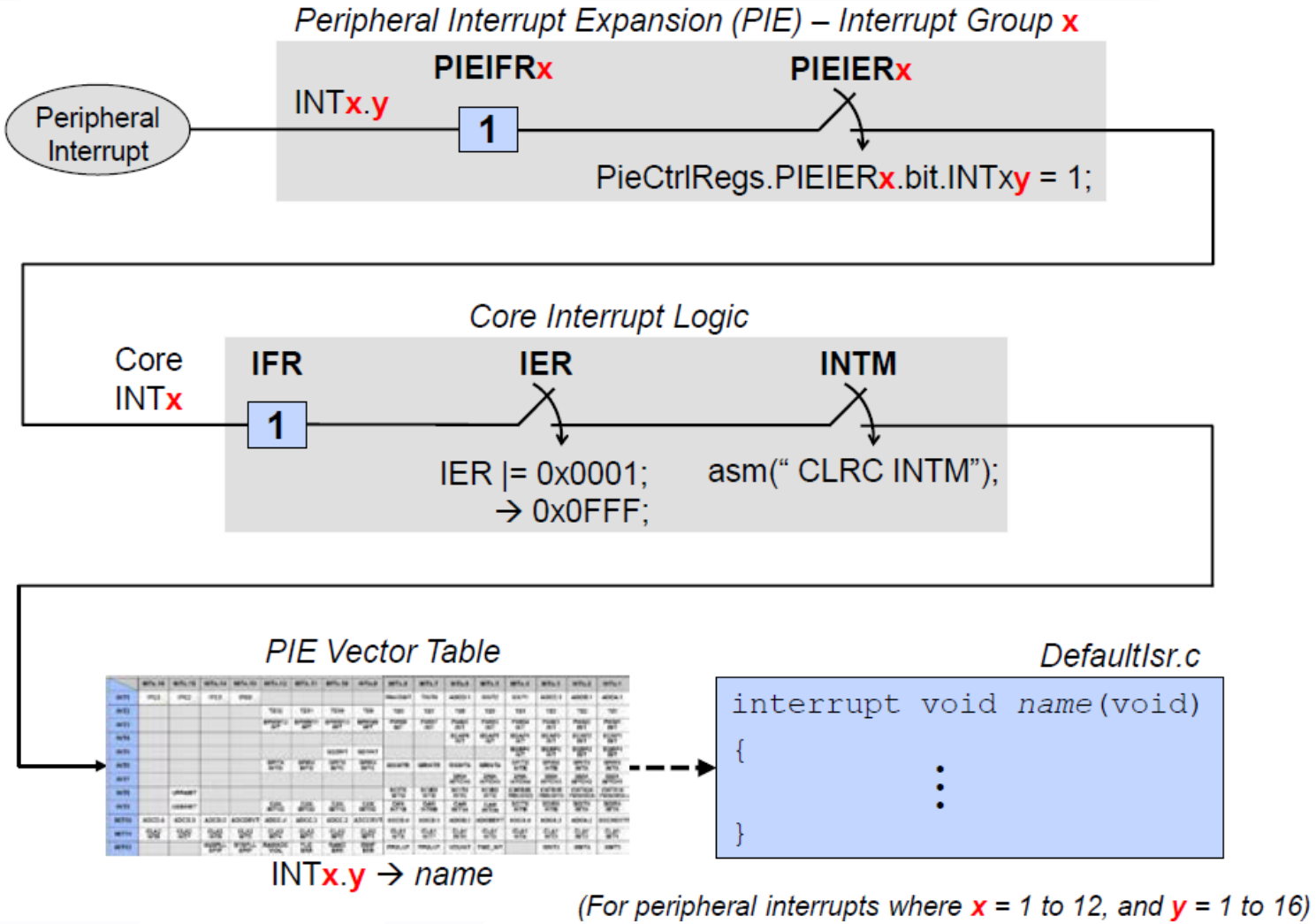
Interrupt Signal Flow– Summary

The following steps occur during an interrupt process:

First, a peripheral interrupt is generated and the PIE interrupt flag register is set.

If the PIE interrupt enable register is enabled, then the core interrupt flag register will be set.

Next, if the core interrupt enable register and global interrupt mask is enabled, the PIE vector table will redirect the code to the interrupt service routine.



Interruptions

Usage : Comment l'utilise t on ?

```
interrupt void adca1_isr(void)
{
   GpioDataRegs.GPBSET.bit.GPIO32 = 1;    // Set
…
…
…
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
   GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;    // Clear
}
```

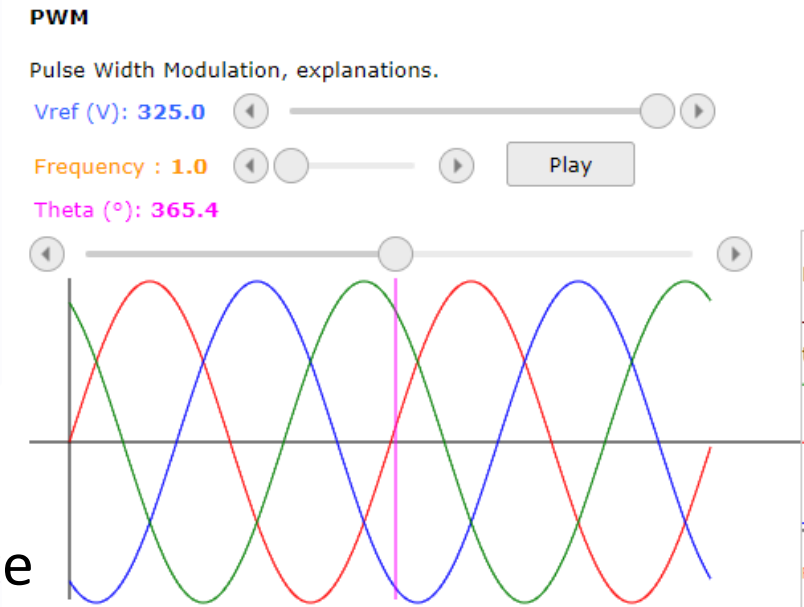
```
void main(void)
{
    // Step 1. Initialize System Control:
    // PLL, WatchDog, enable Peripheral Clocks
    // This example function is found in the F2806x_SysCtrl.c file.
    InitSysCtrl();

    // Step 2. Initalize GPIO:
    // This example function is found in the F2806x_Gpio.c file and
    // illustrates how to set the GPIO to it's default state.
    // InitGpio();  // Skipped for this example

    // Step 3. Clear all interrupts and initialize PIE vector table:
    // Disable CPU interrupts
    DINT;
   InitPieCtrl();
   IER = 0x0000;
   IFR = 0x0000;
   InitPieVectTable();

   EALLOW;
   PieVectTable.ADCA1_INT = &adca1_isr; //function for ADCA interrupt 1
   EDIS;
   PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
   IER |= M_INT1; //Enable group 1 interrupts

   EINT;  // Enable Global interrupt INTM
   ERTM;  // Enable Global realtime interrupt DBGM

    // Step 6. IDLE loop. Just sit and loop forever (optional)
    for(;;) { }
}
```

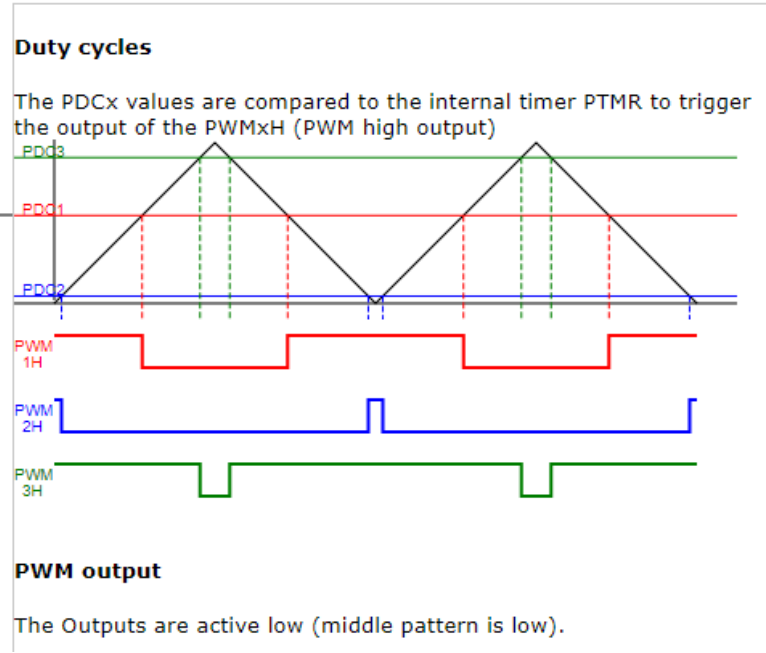
Pulse Width Modulation (PWM)

What : Qu'est ce que c'est ?

- Normal PWM and Motor Control PWM
- Generating multiple, synchronized pulse width modulated outputs
- Module to control chopper and inverter
- Has internal timer
- Use Duty Cycle register to update the outputs on timer crossing their value
- Motor Control PWM have dead band generator, can synchronize the ADC start

http://baghli.com/pwm



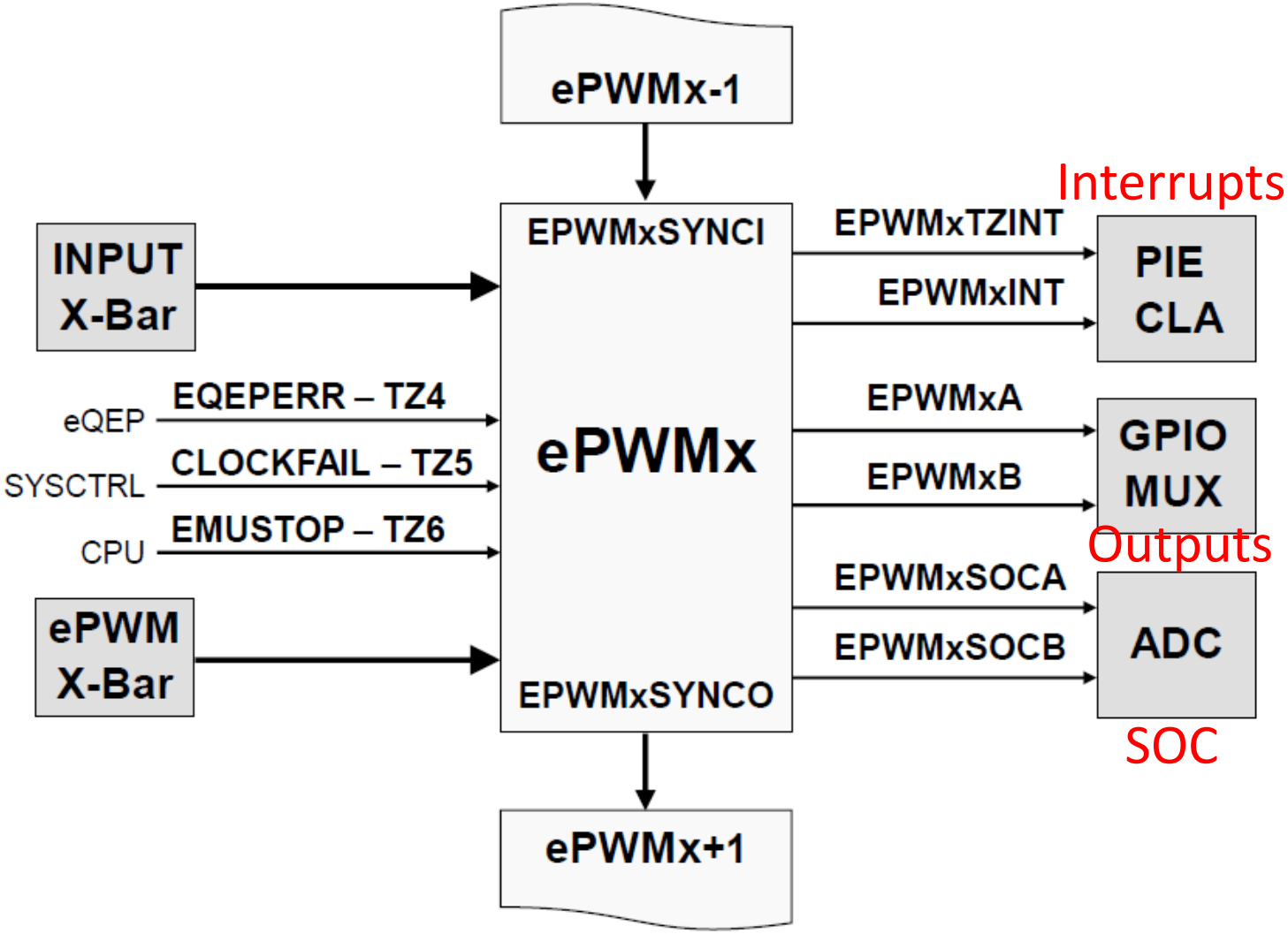


dsPIC30F Family Reference Manual, http://ww1.microchip.com/downloads/en/devicedoc/70046e.pdf
http://baghli.com/pwm/

Pulse Width Modulation (PWM)

How : Comment fonctionne t elle ?

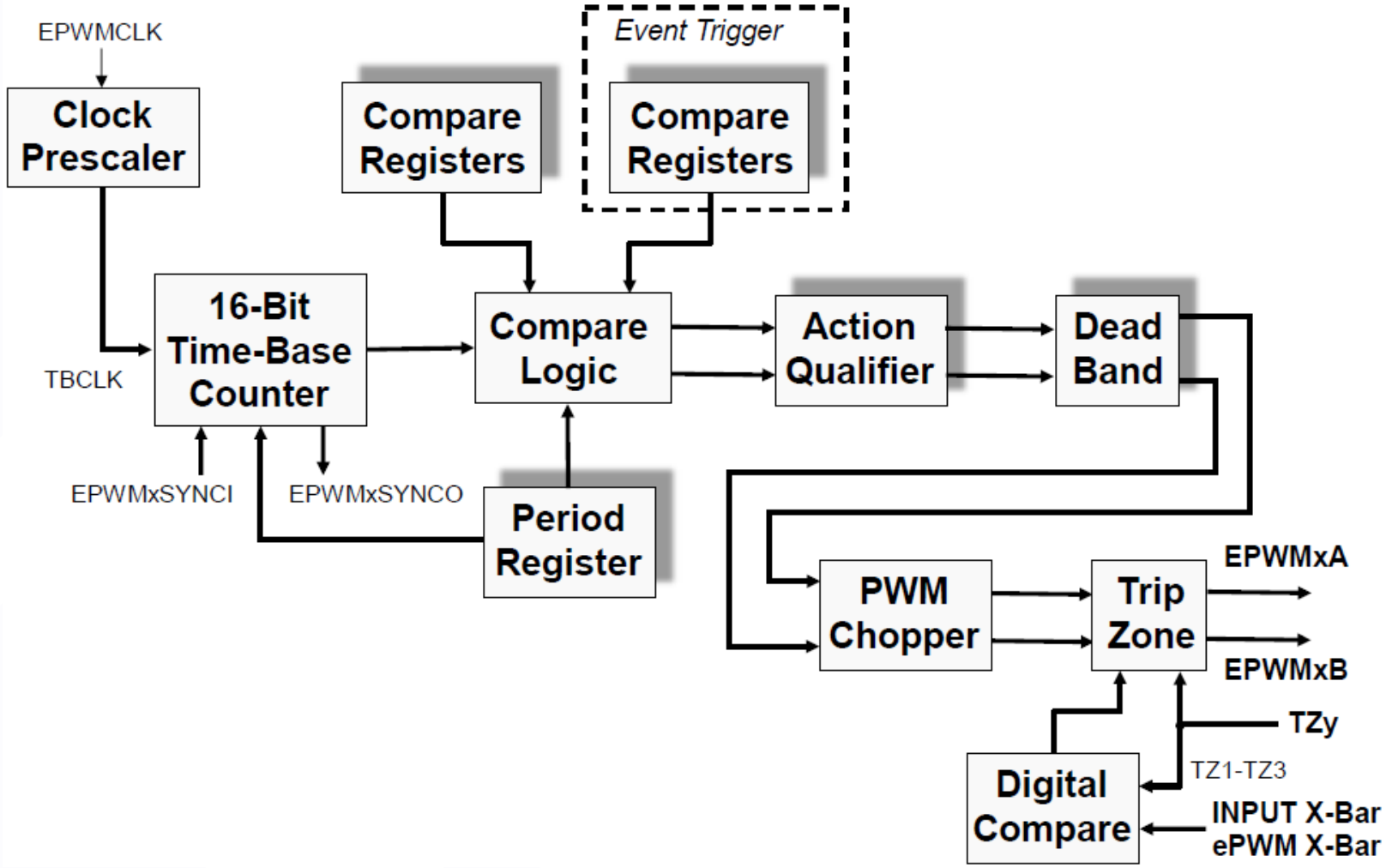
- An ePWM module can be synchronized with adjacent ePWM modules.
- The generated PWM waveforms are available as outputs on the GPIO pins.
- Additionally, the EPWM module can generate ADC starter conversion signals (SOC) and generate interrupts to the PIE block.
- External trip zone signals can trip the output and generate interrupts, too.
- The outputs of the comparators are used as inputs to the digital compare sub-module.



Pulse Width Modulation (PWM)

How : Comment fonctionne t elle ?

- The ePWM, or enhanced PWM block diagram, consists of a series of sub-modules



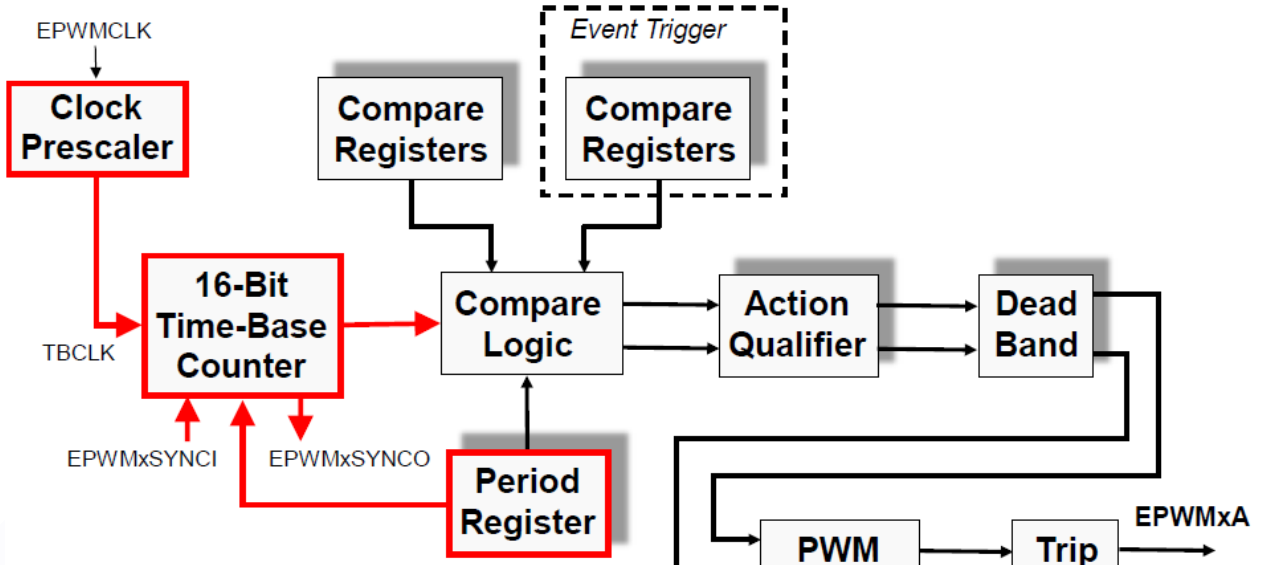
Pulse Width Modulation (PWM)

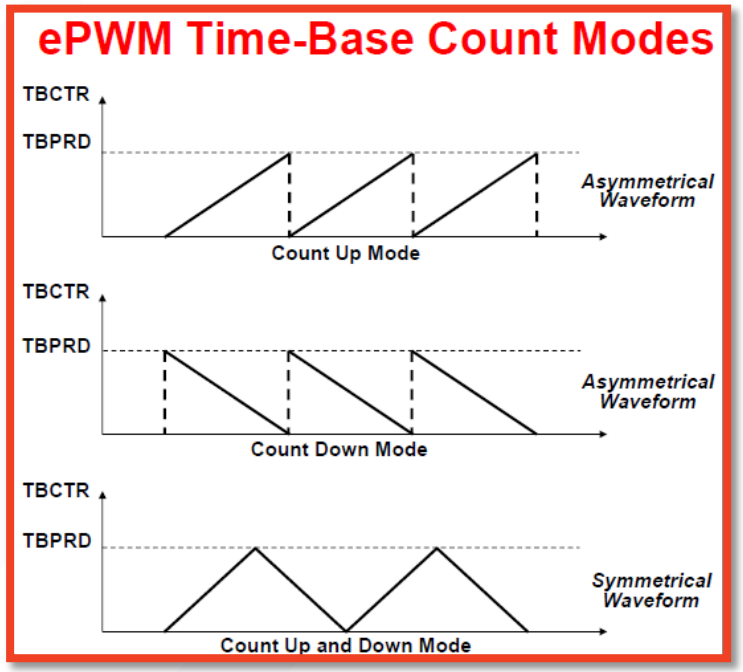
How : Comment fonctionne t elle ?

ePWM Time-Base Sub-Module

In the time-base sub-module, the clock prescaler (TBCTL.CLKDIV) divides down the device core system clock and clocks the 16-bit time-base counter.
The time-base counter (TBCTR) is used to generate asymmetrical and symmetrical waveforms using three different count modes: count-up mode, countdown mode, and count up and down mode (TBCTL.CTRMODE).
A period register is used to control the maximum count value (TBPRD).

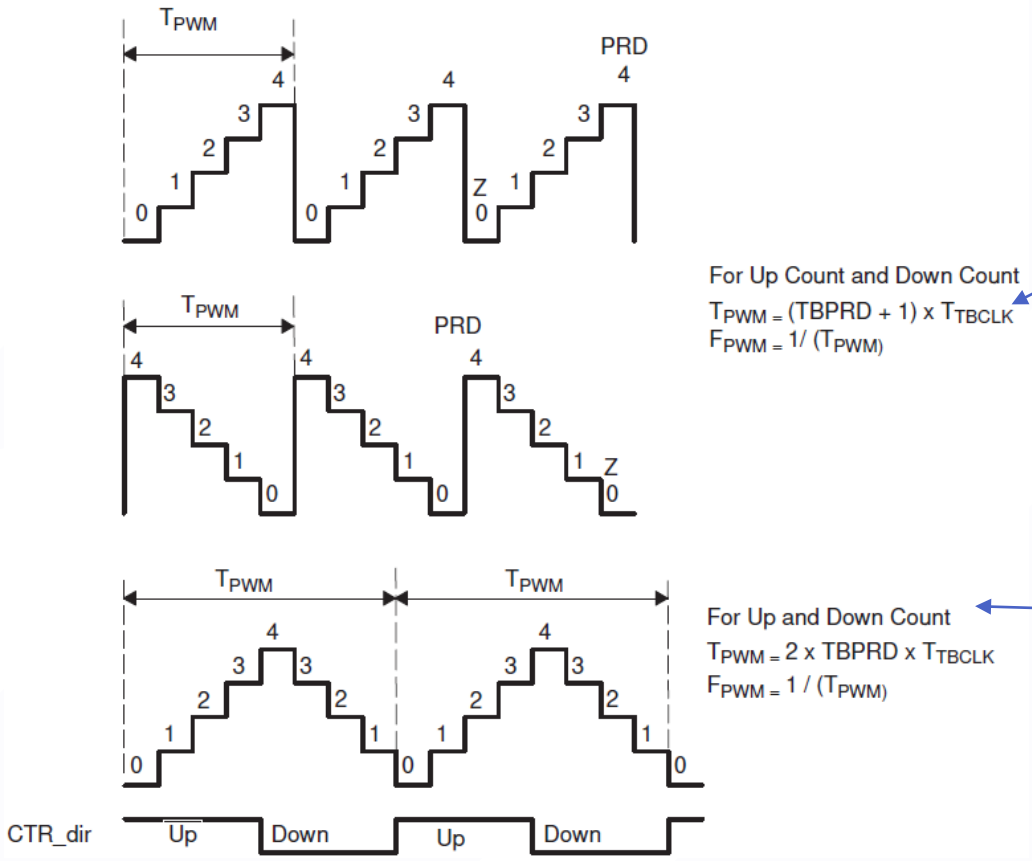
Additionally, the time-base counter has the capability to be synchronized and phase-shifted (TBPHS) with other ePWM units (TBCTL).





Pulse Width Modulation (PWM)

ePWM Time-Base Sub-Module



For Up Count and Down Count

$T_{PWM} = (TBPRD + 1) \times T_{TBCLK}$

$F_{PWM} = 1/(T_{PWM})$

For Up and Down Count

$T_{PWM} = 2 \times TBPRD \times T_{TBCLK}$

$F_{PWM} = 1/(T_{PWM})$

EPwmxRegs.TBCTL

Upper Register:

Phase Direction
0 = count down after sync
1 = count up after sync

TBCLK = EPWMCLK / (HSPCLKDIV * CLKDIV)

15 - 14	13	12 - 10	9 - 7
FREE_SOFT	PHSDIR	CLKDIV	HSPCLKDIV

Emulation Halt Behavior
00 = stop after next CTR inc/dec
01 = stop when:
Up Mode; CTR = PRD
Down Mode; CTR = 0
Up/Down Mode; CTR = 0
1x = free run (do not stop)

TB Clock Prescale
000 = /1 (default)
001 = /2
010 = /4
011 = /8
100 = /16
101 = /32
110 = /64
111 = /128

High Speed TB Clock Prescale
000 = /1
001 = /2 (default)
010 = /4
011 = /6
100 = /8
101 = /10
110 = /12
111 = /14

Lower Register:

Counter Mode
00 = count up
01 = count down
10 = count up and down
11 = stop – freeze (default)

Software Force Sync Pulse
0 = no action
1 = force one-time sync

6	5 - 4	3	2	1 - 0
SWFSYNC	SYNCOSEL	PRDLD	PHSEN	CTRMODE

Sync Output Select
(source of EPWMxSYNC0 signal)
00 = EPWMxSYNCI
01 = CTR = 0
10 = CTR = CMPB *
11 = disable SyncOut

Period Shadow Load
0 = load on CTR = 0
1 = load immediately

Phase Reg. Enable
0 = disable
1 = CTR = TBPHS on EPWMxSYNCI signal

Pulse Width Modulation (PWM)

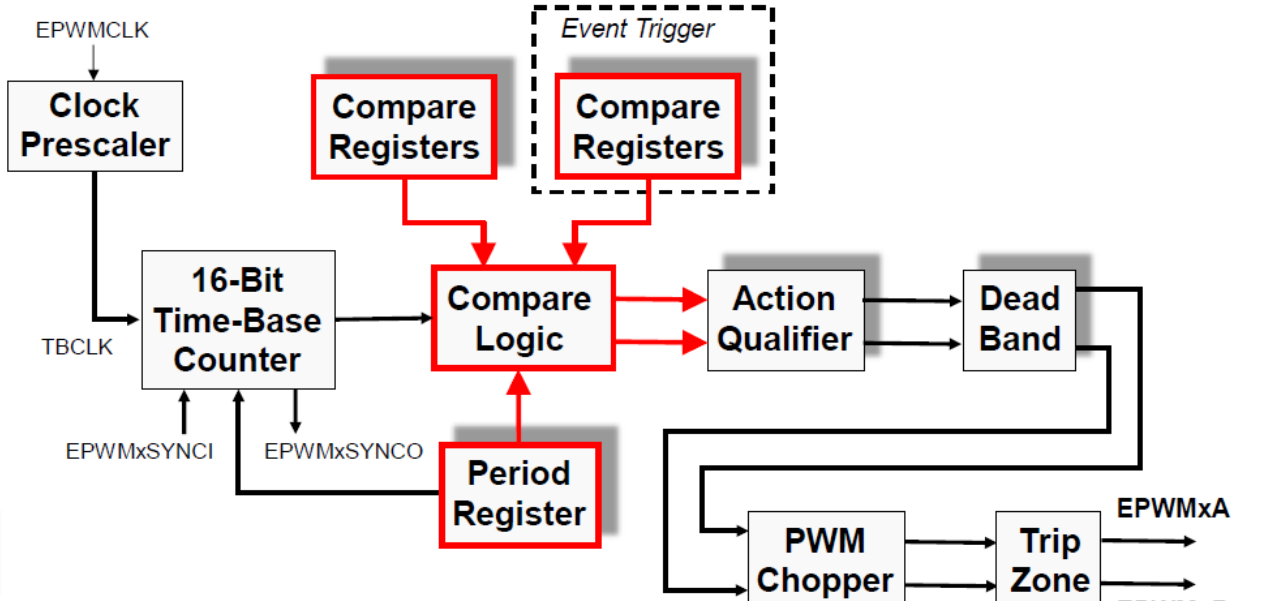
How : Comment fonctionne t elle ?

ePWM Compare Sub-Module

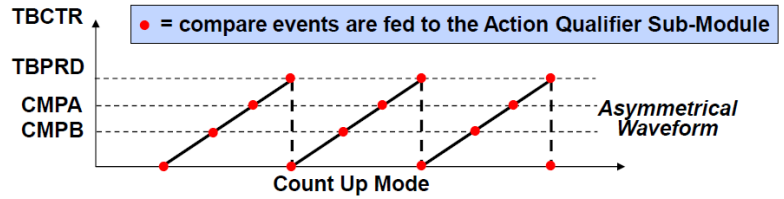
The compare sub-module uses two compare registers (CMPA, CMPB) to detect time-base count (TBCTR) matches.

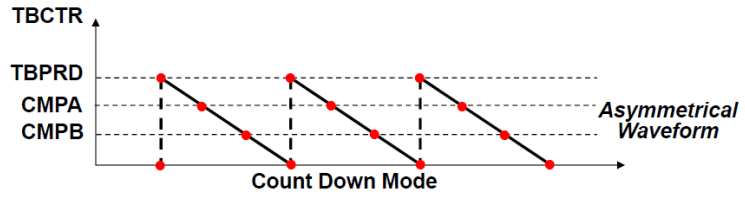
These compare match events are fed into the action qualifier sub-module.

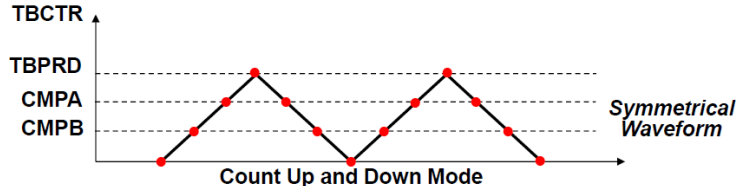
The output of this block feeds two signals into the action qualifier.



ePWM Compare Event Waveforms







Pulse Width Modulation (PWM)

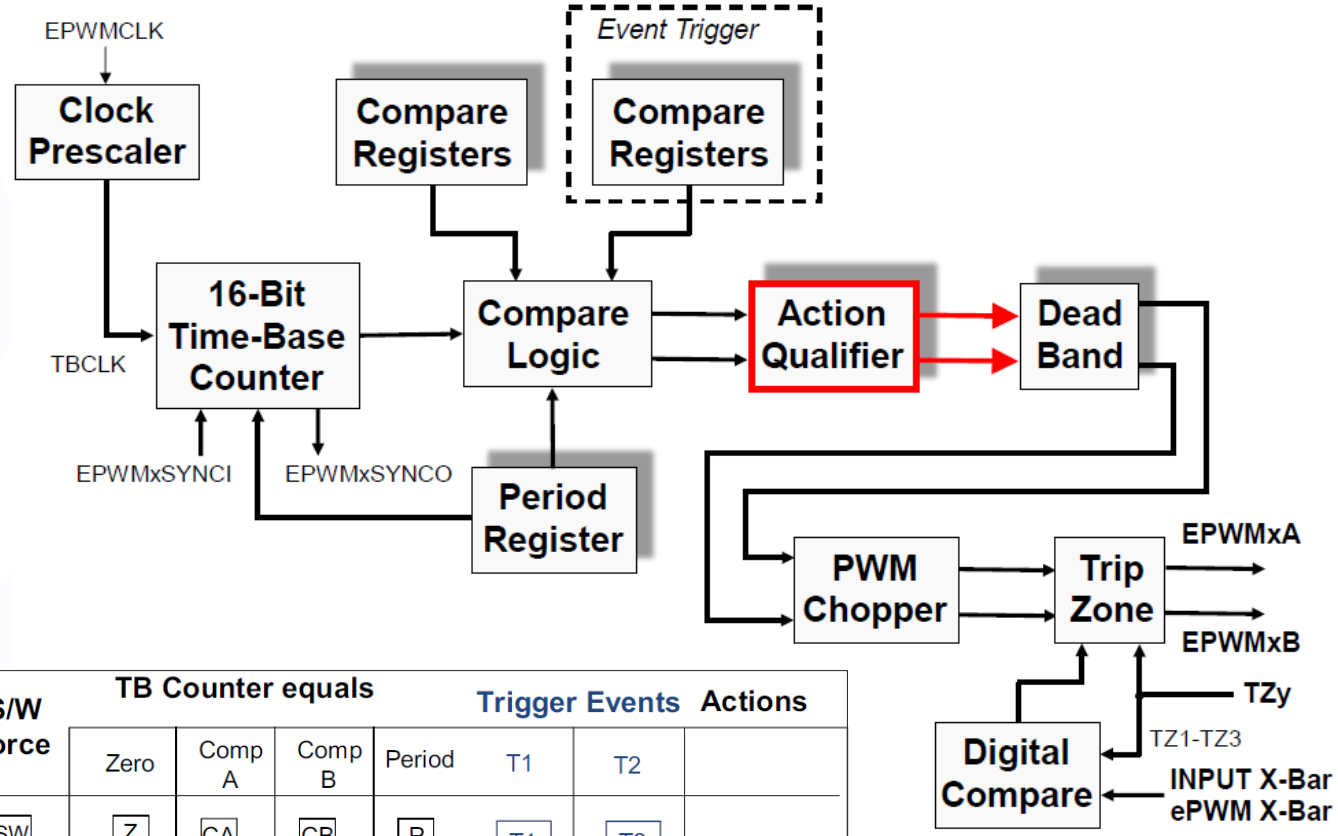
How : Comment fonctionne t elle ?

ePWM Action Qualifier Sub-Module

The action qualifier sub-module uses the inputs from the compare logic and time-base counter to generate various actions on the output pins.

This table shows the various action qualifier compare-match options for when the time-base counter equals zero (Z), compare A match (CAU, CAD), compare B match (CBU, CBD), and period match (P), or a Trigger event (T1, T2) based on a comparator, trip, or sync signal.

Based on the selected match option, the output pins can be configured to do nothing, clear low, set high, or toggle. Also, the output pins can be forced to any action using software.

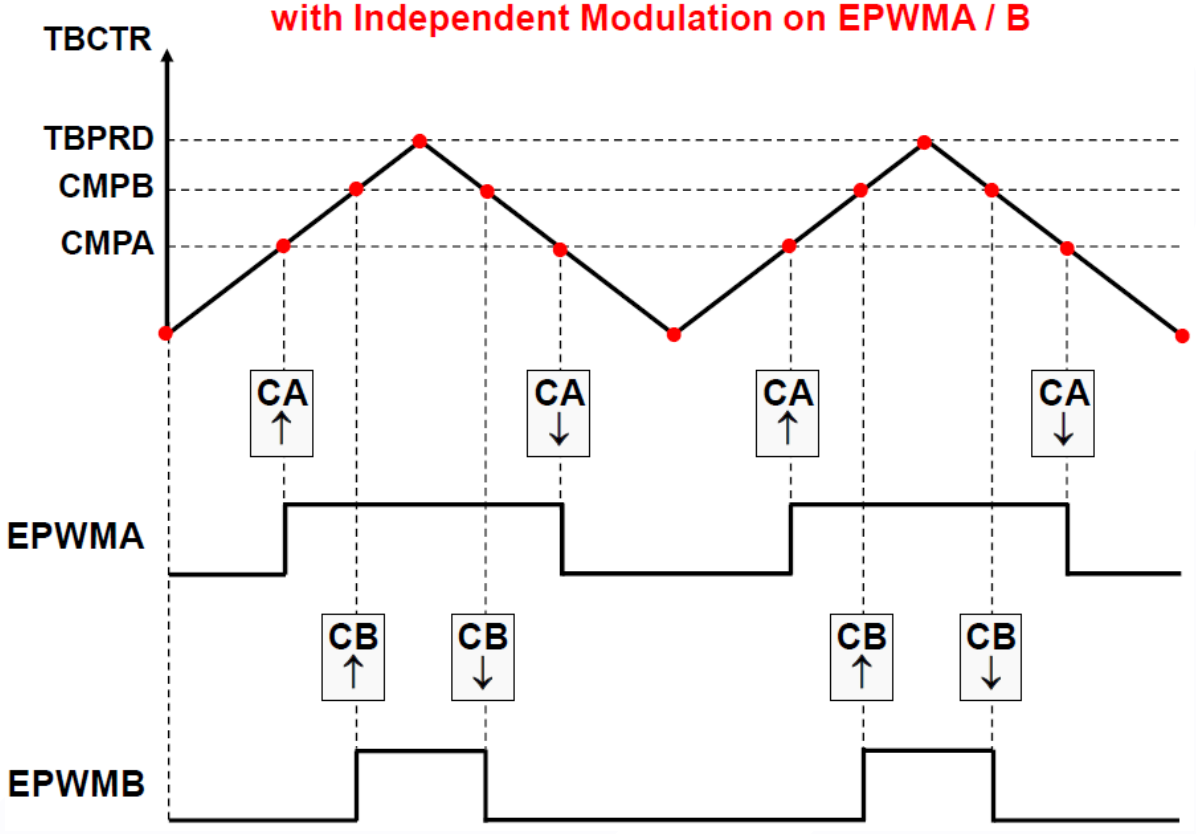


S/W force	TB Counter equals				Trigger Events		Actions
	Zero	Comp A	Comp B	Period	T1	T2	
SW ✕	Z ✕	CA ✕	CB ✕	P ✕	T1 ✕	T2 ✕	Do Nothing
SW ↓	Z ↓	CA ↓	CB ↓	P ↓	T1 ↓	T2 ↓	Clear Lo
SW ↑	Z ↑	CA ↑	CB ↑	P ↑	T1 ↑	T2 ↑	Set Hi
SW	Z	CA	CB	P	T1	T2	Toggle

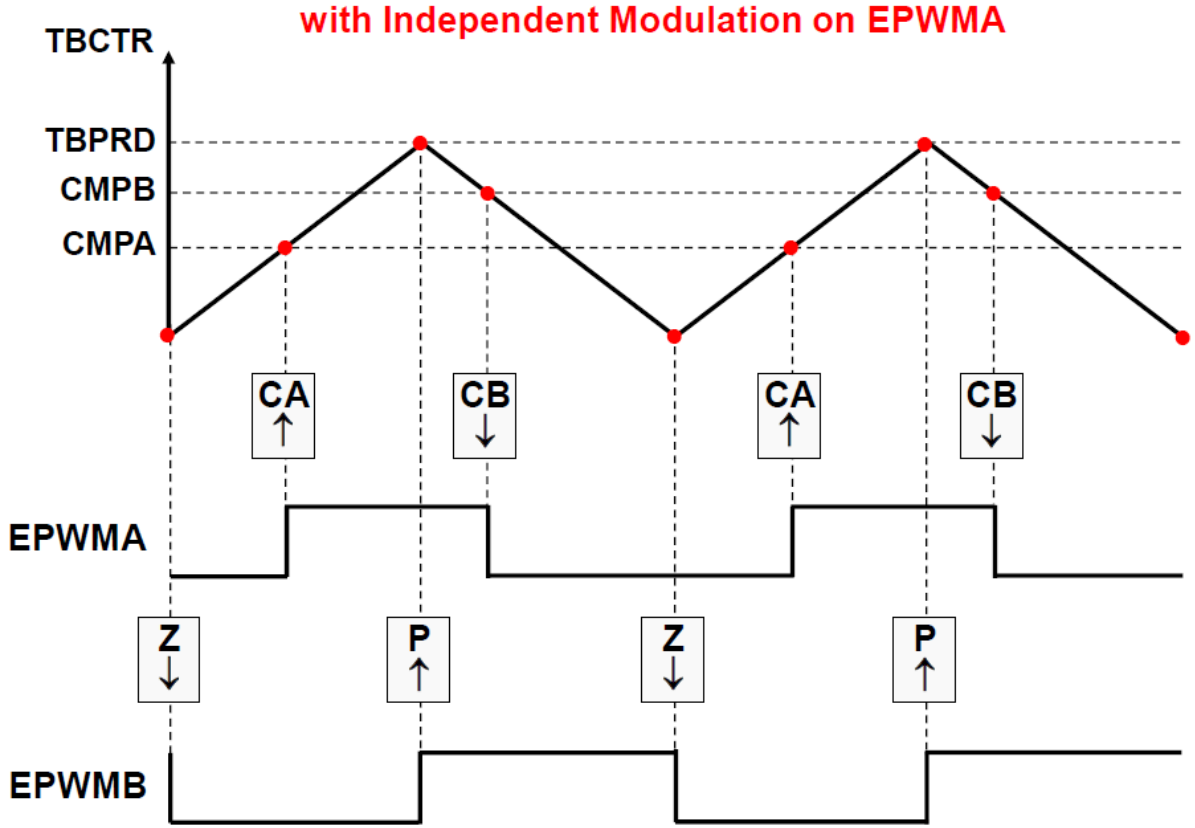
Pulse Width Modulation (PWM)

How : Comment fonctionne t elle ?
Usage : Comment l'utilise t on ?

ePWM Action Qualifier Sub-Module




```
// Set actions
   EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;    // Set PWM1A on event A, up count
   EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;  // Clear PWM1A on event A, down count
   EPwm1Regs.AQCTLB.bit.CBU = AQ_SET;    // Set PWM1B on event B, up count
   EPwm1Regs.AQCTLB.bit.CBD = AQ_CLEAR;  // Clear PWM1B on event B, down count
```




```
// Set actions
   EPwm1Regs.AQCTLA.bit.CAU = AQ_SET
   EPwm1Regs.AQCTLA.bit.CBD = AQ_CLEAR;
   EPwm1Regs.AQCTLB.bit.ZRO = AQ_CLEAR;
   EPwm1Regs.AQCTLB.bit.PRD = AQ_SET;
```

Pulse Width Modulation (PWM)

Usage : Comment l'utilise t on ?

On configure les GPIO pour pointer vers les sorties PWM

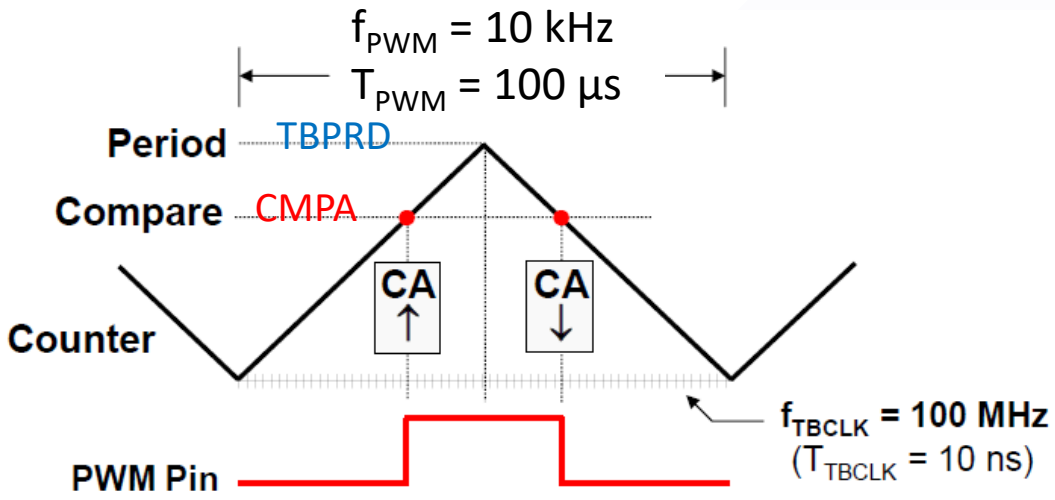
```
void Gpio_setup(void)
{
// Enable PWM1-3 on GPIO0-GPIO5
    EALLOW;
    GpioCtrlRegs.GPAPUD.bit.GPIO0 = 0;    // Enable pullup on GPIO0
    GpioCtrlRegs.GPAPUD.bit.GPIO1 = 0;    // Enable pullup on GPIO1
    GpioCtrlRegs.GPAPUD.bit.GPIO2 = 0;    // Enable pullup on GPIO2
    GpioCtrlRegs.GPAPUD.bit.GPIO3 = 0;    // Enable pullup on GPIO3
    GpioCtrlRegs.GPAPUD.bit.GPIO4 = 0;    // Enable pullup on GPIO4
    GpioCtrlRegs.GPAPUD.bit.GPIO5 = 0;    // Enable pullup on GPIO5
    GpioCtrlRegs.GPAMUX1.bit.GPIO0 = 1;   // GPIO0 = PWM1A
    GpioCtrlRegs.GPAMUX1.bit.GPIO1 = 1;   // GPIO1 = PWM1B
    GpioCtrlRegs.GPAMUX1.bit.GPIO2 = 1;   // GPIO2 = PWM2A
    GpioCtrlRegs.GPAMUX1.bit.GPIO3 = 1;   // GPIO3 = PWM2B
    GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;   // GPIO4 = PWM3A
    GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;   // GPIO5 = PWM3B
    EDIS;
}
```

On configure le module ePWM

```
void InitPWM()
{
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;      // Stop all the TB clocks
    EDIS;
    // setup the Time-Base Period Register (TBPRD)
    // 100MHz/10 kHz/2 = 5000
    EPwm1Regs.TBPRD = 5000;                  // Set timer period
    EPwm1Regs.TBPRD = EPWM_TIMER_TBPRD;   // Set timer period 801 TBCLKs
    EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;   // Phase is 0
    EPwm1Regs.TBCTR = 0x0000;          // Clear counter
    EPwm1Regs.CMPA.bit.CMPA = cmpr;    // Set compare A value
    EPwm1Regs.CMPB.bit.CMPB = cmpr;          // Set Compare B value
    // Setup counter mode
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up and down
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;      // Disable phase loading
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    // Setup shadowing
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
    // Set actions
    EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;  // Set PWM1A on event A, up count
    EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;   // Clear PWM1A on event A, down count
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm1Regs.AQCTLB.bit.CBD = AQ_SET;

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;
}
```

Pulse Width Modulation (PWM)

Usage : Comment l'utilise t on ?



$$TBPRD = \frac{\frac{T_{PWM}}{2}}{T_{TBCLK}} = \frac{f_{TBCLK}}{2f_{PWM}} = \frac{100MHz}{2*10kHz} = 5000$$

CMPA : 0000 → duty cycle : 100%

CMPA : 2500 → duty cycle : 50%

CMPA : 3750 → duty cycle : 33%

CMPA : 5000 → duty cycle : 0%

```
void InitPWM()
{
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;      // Stop all the TB clocks
    EDIS;
    // setup the Time-Base Period Register (TBPRD)
    // 100MHz/10 kHz/2 = 5000
    EPwm1Regs.TBPRD = 5000;                  // Set timer period
    EPwm1Regs.TBPRD = EPWM_TIMER_TBPRD;   // Set timer period 801 TBCLKs
    EPwm1Regs.TBPHS.bit.TBPHS = 0x0000;    // Phase is 0
    EPwm1Regs.TBCTR = 0x0000;          // Clear counter
    EPwm1Regs.CMPA.bit.CMPA = cmpr;    // Set compare A value
    EPwm1Regs.CMPB.bit.CMPB = cmpr;          // Set Compare B value
    // Setup counter mode
    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up and down
    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;      // Disable phase loading
    EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;      // Clock ratio to SYSCLKOUT
    EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
    // Setup shadowing
    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
    EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
    // Set actions
    EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;  // Set PWM1A on event A, up count
    EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;   // Clear PWM1A on event A, down count
    EPwm1Regs.AQCTLB.bit.CBU = AQ_CLEAR;
    EPwm1Regs.AQCTLB.bit.CBD = AQ_SET;

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;
}
```

Pulse Width Modulation (PWM)

Usage : Comment l'utilise t on ?

Pour une utilisation avec des interruptions enclenchées par le module ePWM1 donc : INT3.1

dans main.c

```
...
InitPieVectTable();

EALLOW;  // This is needed to write to EALLOW protected registers
PieVectTable.EPWM1_INT = &epwm_isr;
EDIS;    // This is needed to disable write to EALLOW protected registers

// Enable CPU INT3 which is connected to EPWM1-3 INT
IER |= M_INT3;

// Enable EPWM INTn in the PIE: Group 3 interrupt 1-6
PieCtrlRegs.PIEIER3.bit.INTx1 = 1;

// Enable global Interrupts and higher priority real-time debug events:
EINT;   // Enable Global interrupt INTM
ERTM;   // Enable Global realtime interrupt DBGM
...
```

avec en global

```
int InterruptCount=0;
```

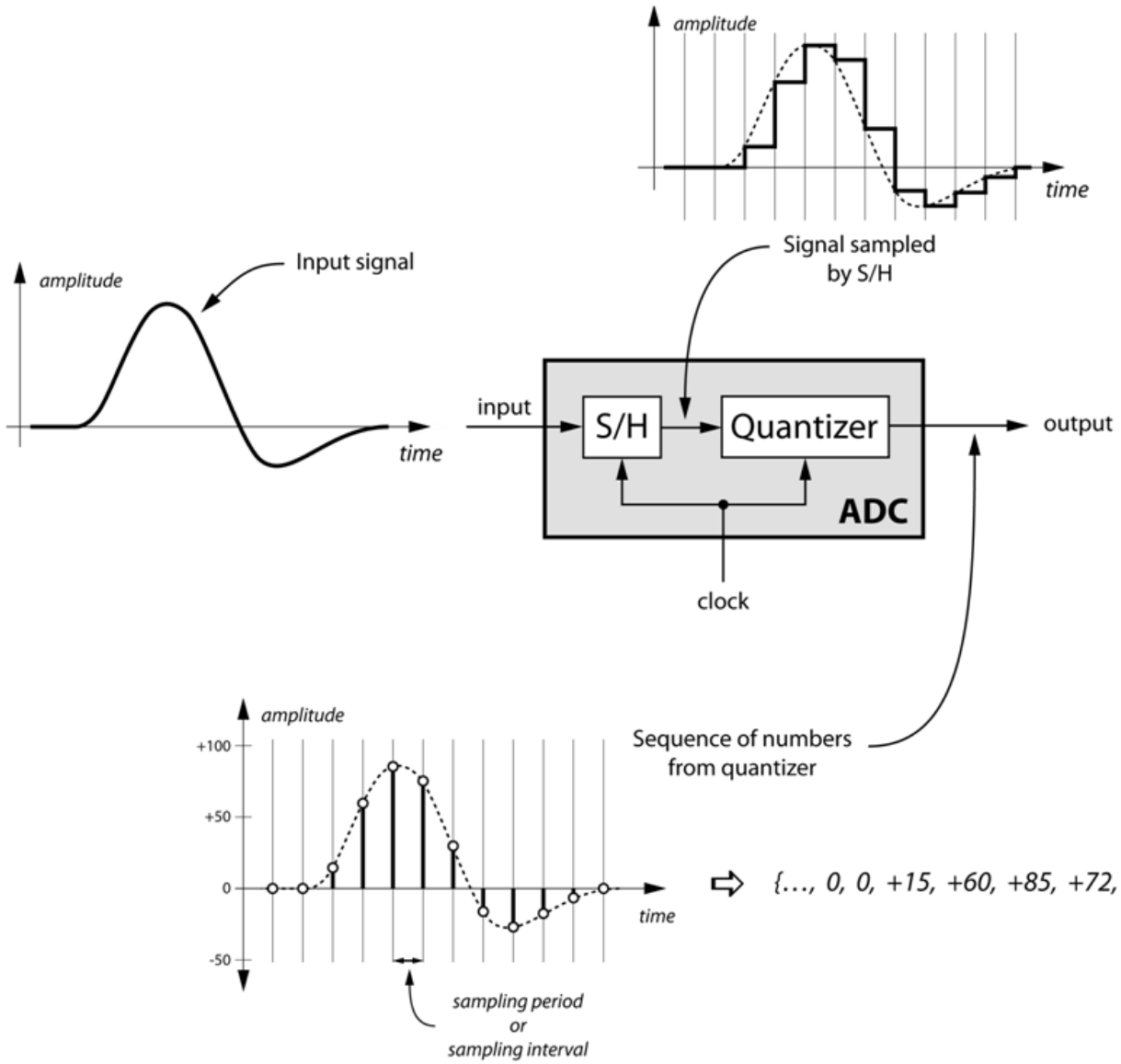
```
// epwm_isr
__interrupt void epwm_isr()
{
    GpioDataRegs.GPASET.bit.GPIO12 = 1;    // Set
    InterruptCount++;
    if (InterruptCount&1)  GpioDataRegs.GPASET.bit.GPIO22 = 1;
            else       GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;  // Clear

    // Clear INT flag
    EPwm1Regs.ETCLR.bit.INT = 1;
    // Acknowledge this interrupt to receive more interrupts from group 3
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;

    GpioDataRegs.GPACLEAR.bit.GPIO12 = 1;  // Clear
}
```

```
void InitPWM()
{ ...
// Interrupt where we will change the Compare Values
    EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO; // Select INT on Zero event
    EPwm1Regs.ETSEL.bit.INTEN = 1;            // Enable INT
    EPwm1Regs.ETPS.bit.INTPRD = ET_1ST;       // Generate INT on 1st event
    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
    EDIS;
}
```

Analog to Digital Converter (ADC)

What : Qu'est ce que c'est ?

- Convert a continuous external voltage (from 0V to 3.3V) to an internal numerical value value (0 to 4095)
- If VDD=3.3V and the ADC resolution is 10 bits, the quantum is $\frac{3.3}{2^{12}} = 0.8\ mV$. It is the smallest voltage value that can be detected by the ADC
- It is also the resolution in Volts



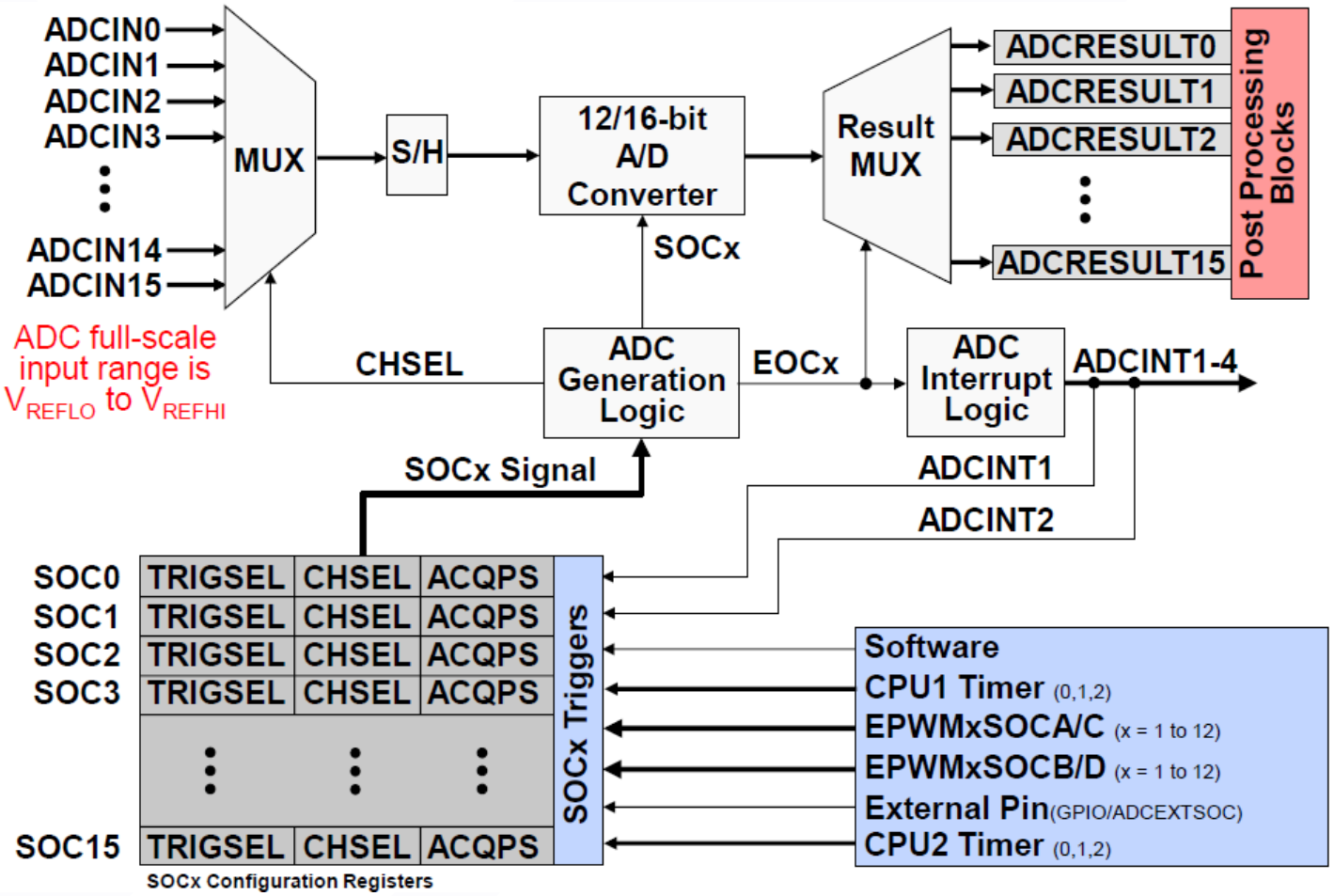
Analog to Digital Converter (ADC)

How : Comment fonctionne t il ?

4 ADC modules. Each module is based around a 12-bit or 16-bit converter.
Differential signal conversions (16-bit mode only)
Single-ended signal conversions (12-bit mode only)

There are 16 input channels and 16 result registers.
The ADC triggering and conversion sequencing is managed by a series of start-of-conversion (SOCx) configuration registers. Each SOCx register configures a single channel conversion. It specifies the trigger source that starts the conversion, the channel to convert, and the acquisition sample window duration.
The triggers include software by selecting a bit, CPU timers 0/1/2, all EPWM, and an external pin.
Additionally, ADCINT 1 and 2 can be fed back for continuous conversions.
The ADC interrupt logic can generate up to nine interrupts. The results for SOC 0 through 15 will appear in result registers 0 through 15.

ADC Module Block Diagram



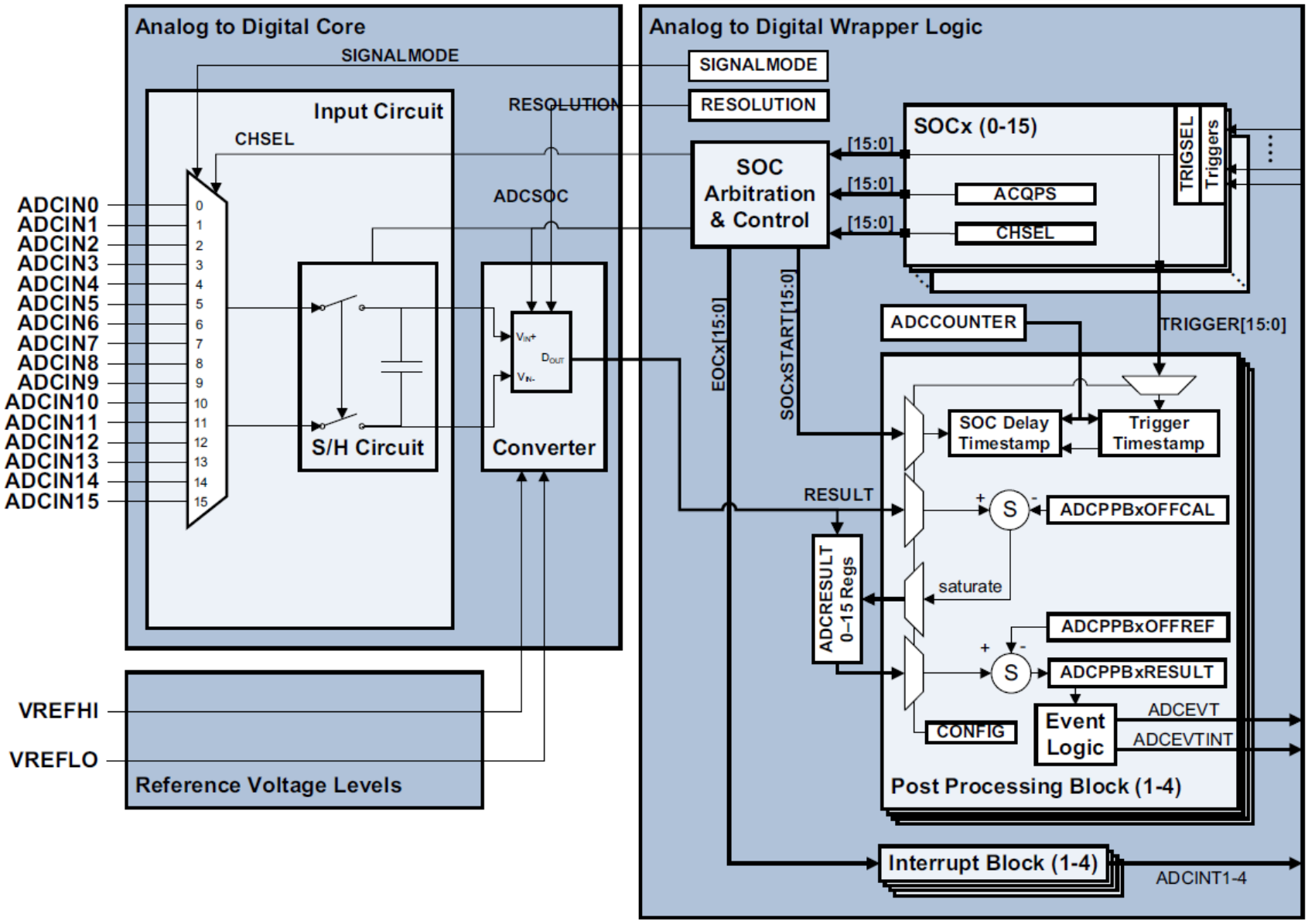
Analog to Digital Converter (ADC)

How :

Comment fonctionne t il ?

The block diagram for the ADC core and the ADC wrapper

ADC Module Block Diagram



Analog to Digital Converter (ADC)

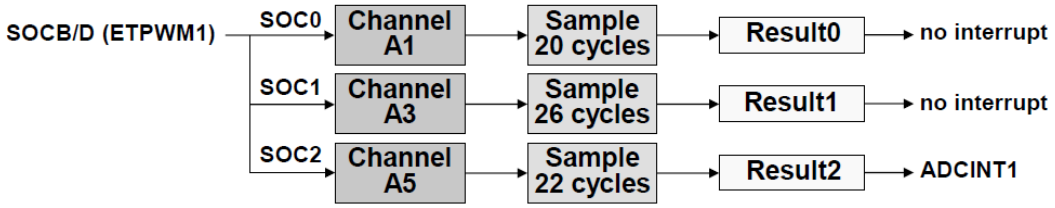
How : Comment fonctionne t il ?

A conceptual view highlighting a single ADC start-of-conversion functional flow from triggering to interrupt generation.

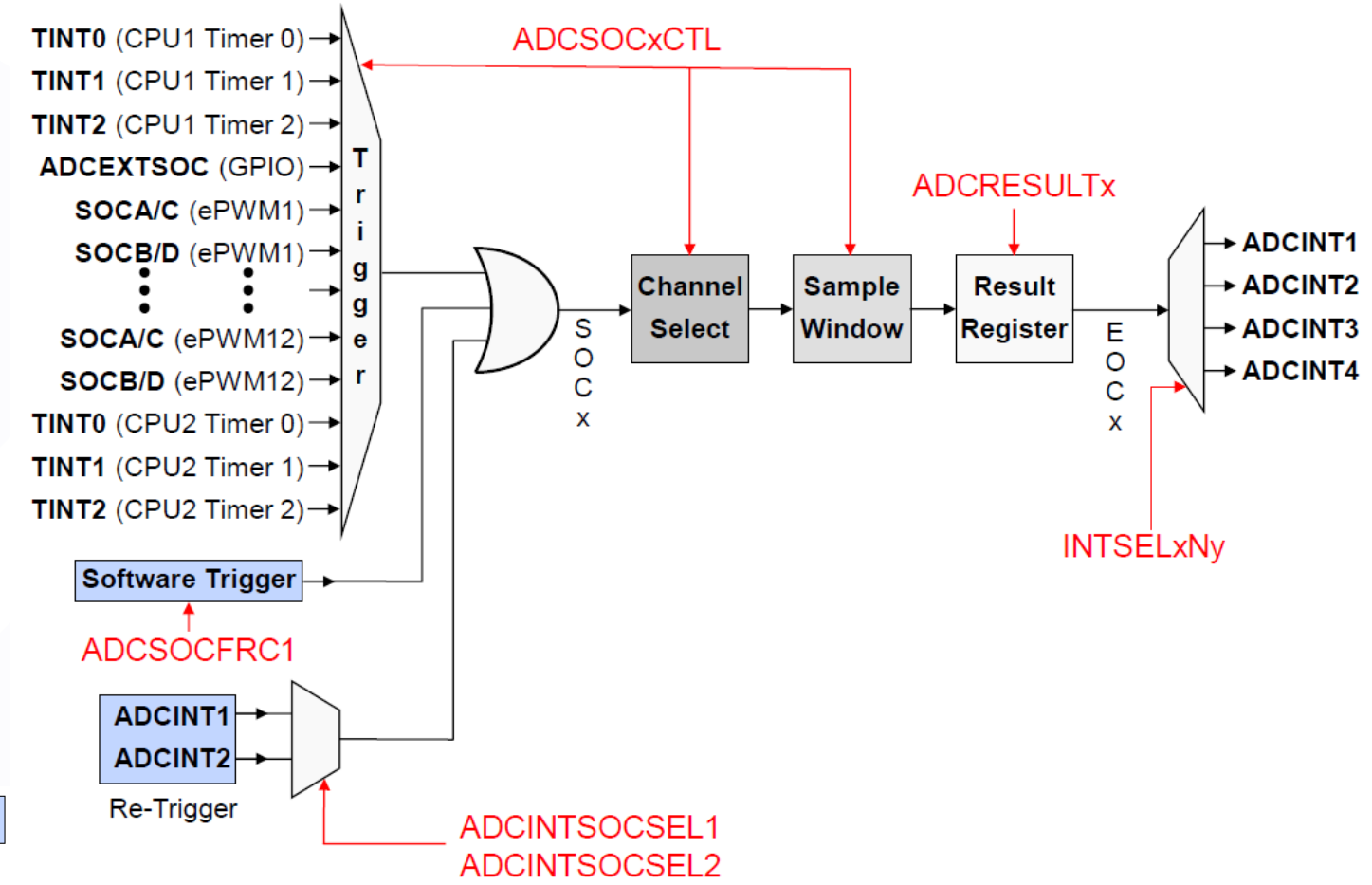
This figure is replicated 16 times and the red text indicates the register names.

Example:

Sample A1 → A3 → A5 when ePWM1 SOCB/D is generated and then generate ADCINT1:



ADC SOCx Functional Diagram



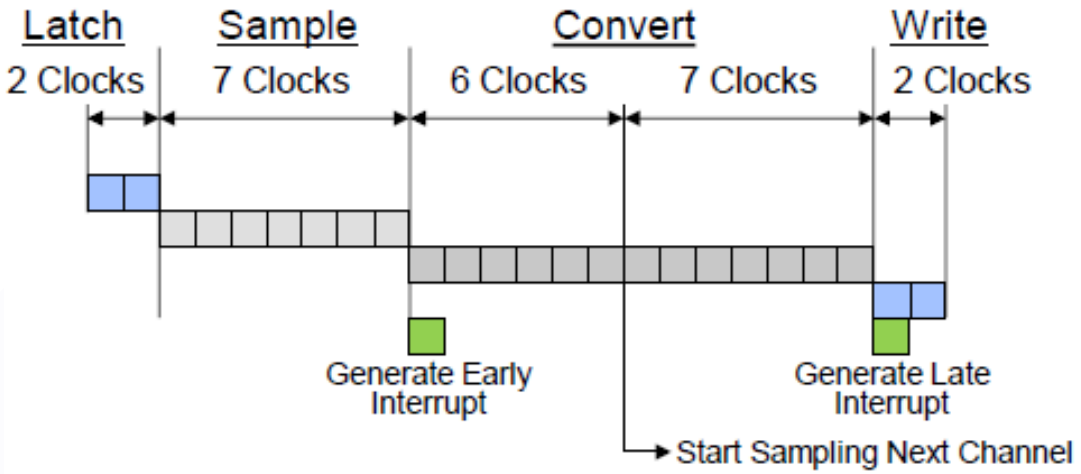
This block diagram is replicated 16 times

Analog to Digital Converter (ADC)

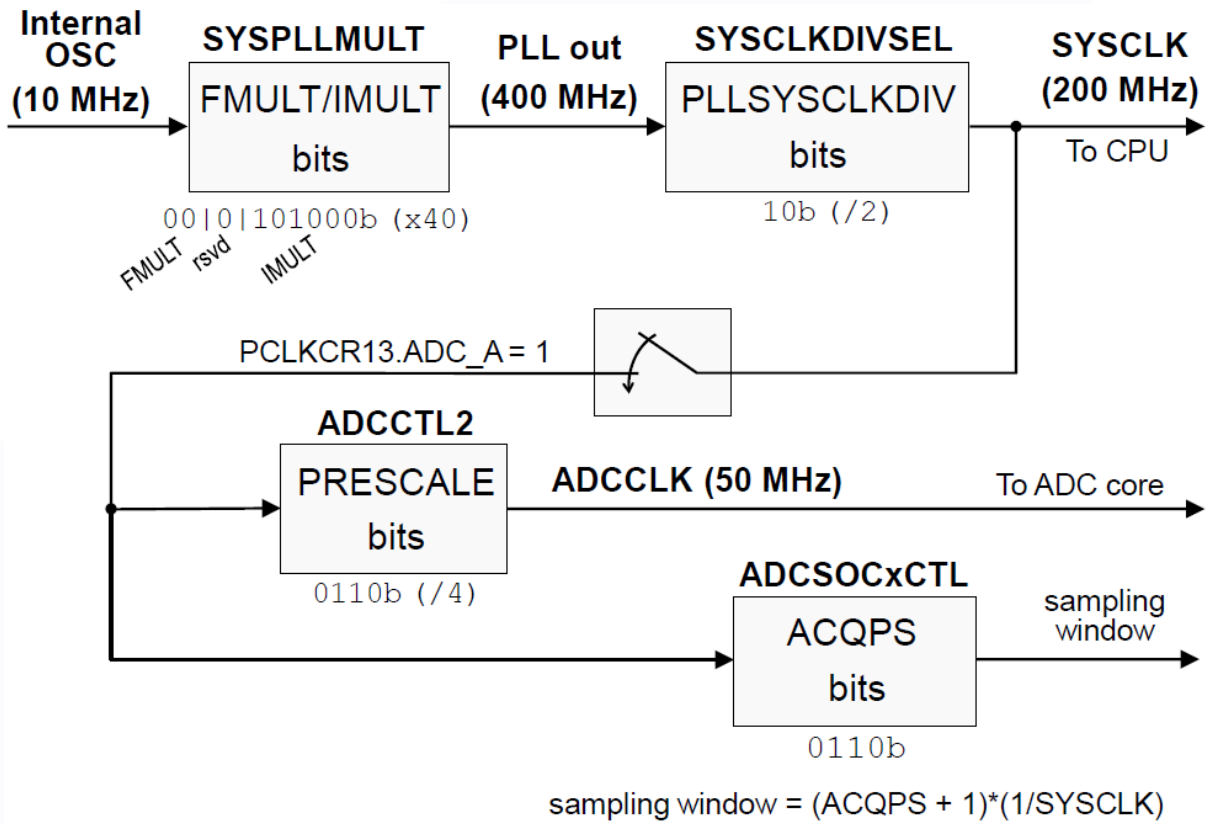
How : Comment fonctionne t il ?

Il faut laisser le temps à l'échantillonneur et au convertisseur pour finir la conversion
7 cycles = 6 (à indiquer) + 1
`AdcRegs.ADCSOC0CTL.bit.ACQPS = 6`

ADC Timing: Sequential sampling



ADC Clocking Flow



Note: Sampling window of 7 cycles is minimum and it can be larger

Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

- ADCTRL1 (Control Register 1)
  - module reset, ADC enable
  - busy/busy channel
  - reference select
  - Interrupt generation control
- ADCSOCxCTL(SOC0 to SOC15 Control Registers)
  - trigger source
  - channel
  - acquisition sampling window
- ADCINTSOCSELx(Interrupt SOC Selection 1 and 2 Registers)
  - selects ADCINT1 / ADCINT2 trigger for SOCx
- INTSELxNy(Interrupt x and y Selection Registers)
  - EOC0 –EOC15 source select for ADCINT1-9
- ADCRESULTx (ADC Result 0 to 15 Registers)

ADC Registers

RegistersAdc*z*Regs.*register*** *(z = a, b, c, or d)*

Register	Description
ADCCTL1	Control 1 Register
ADCCTL2	Control 2 Register
ADCSOCxCTL	SOC0 to SOC15 Control Registers
ADCINTSOCSELx	Interrupt SOC Selection 1 and 2 Registers
INTSELxNy	Interrupt x and y Selection Registers
SOCPRICTL	SOC Priority Control Register
ADCBURSTCTL	SOC Burst Control Register
ADCOFFTRIM	Offset Trim Register
ADCRESULTx	ADC Result 0 to 15 Registers

Note: ADCRESULTx header file coding is AdczResultRegs.ADCRESULTx (*not in AdczRegs*)

Refer to the Technical Reference Manual for a complete listing of registers

Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

```
//Set pulse positions to late
AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
//power up the ADC
AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
//delay for 1ms to allow ADC time to power up
DELAY_US(1000);
```

ADC Control Register 1

AdczRegs.ADCCTL1 *(z = a, b, c, or d)*

ADC Busy
0 = ADC available
1 = ADC busy

ADC Busy Channel
When ADCBSY =
0: last channel converted
1: channel currently processing

INT Pulse Generation Control
0 = beginning of conversion
1 = one cycle prior to result

15 - 14	13	12	11 - 8	7	6 - 3	2	1 - 0
reserved	ADCBSY	reserved	ADCBSYCHN	ADCPWDNZ	reserved	INTPULSEPOS	reserved

00h = ADCIN0
01h = ADCIN1
02h = ADCIN2
03h = ADCIN3
04h = ADCIN4
05h = ADCIN5
06h = ADCIN6
07h = ADCIN7

08h = ADCIN9
09h = ADCIN10
0Ah = ADCIN11
0Bh = ADCIN12
0Ch = ADCIN13
0Dh = ADCIN14
0Eh = ADCIN15
0Fh = ADCIN16

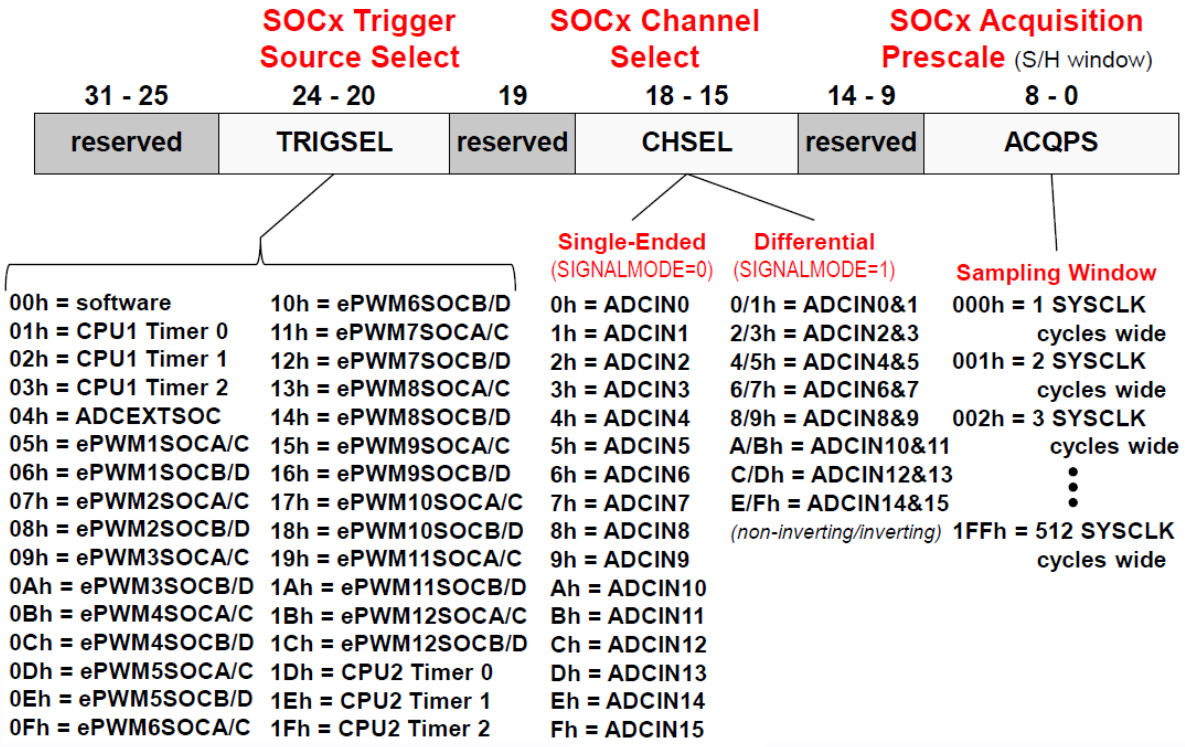
ADC Power Down
Analog circuitry is:
0 = powered down
1 = powered up

Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

ADC SOC0 –SOC15 Control Registers

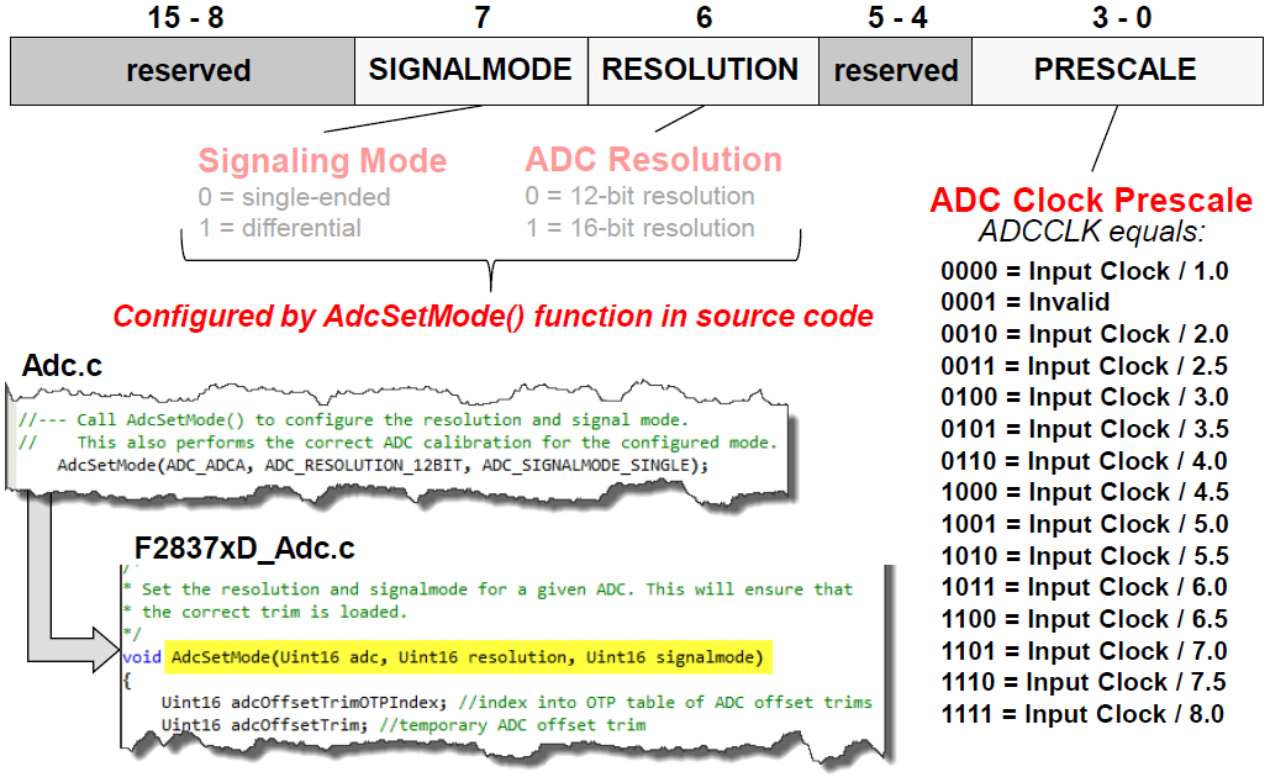
AdczRegs. ADCSOCxCTL *(z = a, b, c, or d)*




```
// ADCINC2 : courant de la phase a : Ia
  AdccRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin C2
  AdccRegs.ADCSOC0CTL.bit.ACQPS = 30; //sample window is 100 SYSCLK cycles
  AdccRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
```

ADC Control Register 2

AdczRegs.ADCCTL2 *(z = a, b, c, or d)*



Configured by AdcSetMode() function in source code

Adc.c

```
//--- Call AdcSetMode() to configure the resolution and signal mode.
//    This also performs the correct ADC calibration for the configured mode.
   AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
```

F2837xD_Adc.c

```
/*
* Set the resolution and signalmode for a given ADC. This will ensure that
* the correct trim is loaded.
*/
void AdcSetMode(Uint16 adc, Uint16 resolution, Uint16 signalmode)
{
   Uint16 adcOffsetTrimOTPIndex; //index into OTP table of ADC offset trims
   Uint16 adcOffsetTrim; //temporary ADC offset trim
```

Definitions for selecting ADC signaling mode and resolution defined in F2837xD_Adc_defines.h

```
AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
```

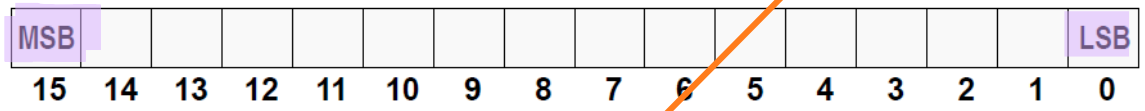
Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

ADC Conversion Result Registers

16-bit Mode

AdcnResultRegs.ADCRESULTx *(n = a-d, x = 0-15)*



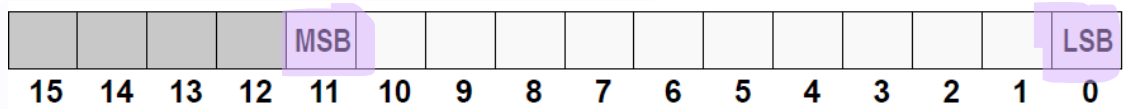
ADCIN*x*P Voltage	ADCIN*x*N Voltage	Digital Results	Adc*n*ResultRegs. ADCRESULT*x*
3.0V	0V	FFFFh	1111|1111|1111|1111
1.5V	1.5V	7FFFh	0111|1111|1111|1111
45µV	3.0V - 45µV	1h	0000|0000|0000|0001
0V	3.0V	0h	0000|0000|0000|0000

- **Differential – two input pins (ADCIN*x*P & ADCIN*x*N)**
- **Input voltage is the difference between the two pins**
- **External reference (VREFHI and VREFLO)**

ADC Conversion Result Registers

12-bit Mode

AdcnResultRegs.ADCRESULTx *(n = a-d, x = 0-15)*



ADCIN*x* Voltage	Digital Results	Adc*n*ResultRegs. ADCRESULT*x*
3.0V	FFFh	0000|1111|1111|1111
1.5V	7FFh	0000|0111|1111|1111
0.00073V	1h	0000|0000|0000|0001
0V	0h	0000|0000|0000|0000

- **Single-ended – one input pin (ADCIN*x*)**
- **External reference (VREFHI and VREFLO)**

```
ADC_Ia = AdccResultRegs.ADCRESULT0;
ADC_Ib = AdcbResultRegs.ADCRESULT0;
ADC_Ic = AdcaResultRegs.ADCRESULT0;
```

Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

ADC Interrupt Trigger SOC Select

Registers 1 & 2

AdczRegs. ADCINTSOCSELx *(z = a, b, c, or d)*

ADCINTSOCSEL2

15 - 14	13 - 12	11 - 10	9 - 8	7 - 6	5 - 4	3 - 2	1 - 0
SOC15	SOC14	SOC13	SOC12	SOC11	SOC10	SOC9	SOC8

ADCINTSOCSEL1

15 - 14	13 - 12	11 - 10	9 - 8	7 - 6	5 - 4	3 - 2	1 - 0
SOC7	SOC6	SOC5	SOC4	SOC3	SOC2	SOC1	SOC0

SOCx ADC Interrupt Select

Selects which, if any, ADCINT triggers SOCx

00 = no ADCINT will trigger SOCx (TRIGSEL field determines SOCx trigger)
01 = ADCINT1 will trigger SOCx (TRIGSEL field ignored)
10 = ADCINT2 will trigger SOCx (TRIGSEL field ignored)
11 = invalid selection

Interrupt Select x and y Register

AdczRegs.INTSELxNy *(z = a, b, c, or d)*

Where x/y = 1/2, 3/4

15	14	13	12	11 - 8
reserved	INTyCONT	INTyE	reserved	INTySEL

7	6	5	4	3 - 0
reserved	INTxCONT	INTxE	reserved	INTxSEL

ADCINTx/y Continuous Mode Enable
0 = one-shot pulse generated (until flag cleared by user)
1 = pulse generated for each EOC

ADCINTx/y Interrupt Enable
0 = disable
1 = enable

ADCINTx/y EOC Source Select
00h = EOC0 is trigger for ADCINTx/y
01h = EOC1 is trigger for ADCINTx/y
02h = EOC2 is trigger for ADCINTx/y
03h = EOC3 is trigger for ADCINTx/y
04h = EOC4 is trigger for ADCINTx/y
05h = EOC5 is trigger for ADCINTx/y
06h = EOC6 is trigger for ADCINTx/y
07h = EOC7 is trigger for ADCINTx/y
08h = EOC8 is trigger for ADCINTx/y
09h = EOC9 is trigger for ADCINTx/y
0Ah = EOC10 is trigger for ADCINTx/y
0Bh = EOC11 is trigger for ADCINTx/y
0Ch = EOC12 is trigger for ADCINTx/y
0Dh = EOC13 is trigger for ADCINTx/y
0Eh = EOC14 is trigger for ADCINTx/y
0Fh = EOC15 is trigger for ADCINTx/y

```
AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 2; //end of SOC2 will set INT1 flag
AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
```

Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

```
ConfigureADC();
InitMyADC();
```

- Initialiser l'ADC
- Configurer l'ADC : Utilisateur

J1.01 +3.3V	J3.21 +5V	J4.40 PWM1A PWMHA	J2.20 GND
J1.02 P32 Info	J3.20 GND	J4.39 PWM1B PWMLA	J2.19 P61
J1.03 P19 FAULT (in)	J3.23 ADCIN14 Va	J4.38 PWM2A PWMHB	J2.18 P123 SCS
J1.04 P18 HallA	J3.24 ADCINC3 Vb	J4.37 PWM2B PWMLB	J2.17 P122
J1.05 P67 HallB	J3.25 ADCINB3 Vc	J4.36 PWM3A PWMHC	J2.16 RESET
J1.06 P111 HallC	J3.26 ADCINA3 Vdc	J4.35 PWM3B PWMLC	J2.15 P58 MOSI
J1.07 P60 SCLK	J3.27 ADCINC2 Ia	J4.34 P24	J2.14 P59 MISO
J1.08 P22	J3.28 ADCINB2 Ib	J4.33 P16	J2.13 P124 ENGATE
J1.09 P105 SCL_OLED	J3.29 ADCINA2 Ic	J4.32 DAC1 BNC1	J2.12 P125 WAKE
J1.10 P104 SDA_OLED	J3.30 ADCINA0	J4.31 DAC2 BNC2	J2.11 P29

```
void ConfigureADC(void)
{
  EALLOW;
  //write configurations
  AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
  AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
  AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4

  AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
  AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
  AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE);
  //Set pulse positions to late
  AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
  AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
  AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
  //power up the ADC
  AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
  AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
  AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
  //delay for 1ms to allow ADC time to power up
  DELAY_US(1000);
  EDIS;
}
```

Analog to Digital Converter (ADC

Usage : Comment l'utilise t on ?

```
ConfigureADC();
InitMyADC();
```

- Initialiser l'ADC
- Configurer l'ADC : Utilisateur

J1.01 +3.3V	J3.21 +5V	J4.40 PWM1A PWMHA	J2.20 GND
J1.02 P32 Info	J3.20 GND	J4.39 PWM1B PWMLA	J2.19 P61
J1.03 P19 FAULT (in)	J3.23 ADCIN14 Va	J4.38 PWM2A PWMHB	J2.18 P123 SCS
J1.04 P18 HallA	J3.24 ADCINC3 Vb	J4.37 PWM2B PWMLB	J2.17 P122
J1.05 P67 HallB	J3.25 ADCINB3 Vc	J4.36 PWM3A PWMHC	J2.16 RESET
J1.06 P111 HallC	J3.26 ADCINA3 Vdc	J4.35 PWM3B PWMLC	J2.15 P58 MOSI
J1.07 P60 SCLK	J3.27 ADCINC2 Ia	J4.34 P24	J2.14 P59 MISC
J1.08 P22	J3.28 ADCINB2 Ib	J4.33 P16	J2.13 P124 ENC
J1.09 P105 SCL_OLED	J3.29 ADCINA2 Ic	J4.32 DAC1 BNC1	J2.12 P125 WAK
J1.10 P104 SDA_OLED	J3.30 ADCINA0	J4.31 DAC2 BNC2	J2.11 P29

```
void InitMyADC()
{
  Uint16 acqps = 30; //75ns  for ADC_RESOLUTION_12BIT
  // Determine minimum acquisition window (in SYSCLKS) based on resolution
//Select the channels to convert and end of conversion flag
  EALLOW;
// ADCINC2 Ia
  AdccRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin C2
  AdccRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdccRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
// ADCINB2 Ib
  AdcbRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin B2
  AdcbRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdcbRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
// ADCINA2 Ic
  AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2;  //SOC0 will convert pin A2
  AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
// ADCIN14 Va
  AdcaRegs.ADCSOC1CTL.bit.CHSEL = 14;  //SOC1 will convert pin A14
  AdcaRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
// ADCINC3 Vb
  AdccRegs.ADCSOC1CTL.bit.CHSEL = 3;  //SOC1 will convert pin C3
  AdccRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdccRegs.ADCSOC1CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
// ADCINB3 Vc
  AdcbRegs.ADCSOC1CTL.bit.CHSEL = 3;  //SOC1 will convert pin B3
  AdcbRegs.ADCSOC1CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdcbRegs.ADCSOC1CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
// ADCINA3 Vdc
  AdcaRegs.ADCSOC2CTL.bit.CHSEL = 3;  //SOC2 will convert pin A3
  AdcaRegs.ADCSOC2CTL.bit.ACQPS = acqps; //sample window is 100 SYSCLK cycles
  AdcaRegs.ADCSOC2CTL.bit.TRIGSEL = 5; //trigger on ePWM1 SOCA/C
```

```
  AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 2; //end of SOC2 will set INT1 flag
  AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;   //enable INT1 flag
  AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
  // ePWM1 SOC à chaque Periode
  EPwm1Regs.ETSEL.bit.SOCAEN = 1;              // Enable SOC on A group
  EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO; // SOC on time-base counter equal to zero (TBCTR = zero)
  EPwm1Regs.ETPS.bit.SOCAPRD  = ET_1ST;  // Generate pulse on 1st event
  EDIS;
}
```

Analog to Digital Converter (ADC)

Usage : Comment l'utilise t on ?

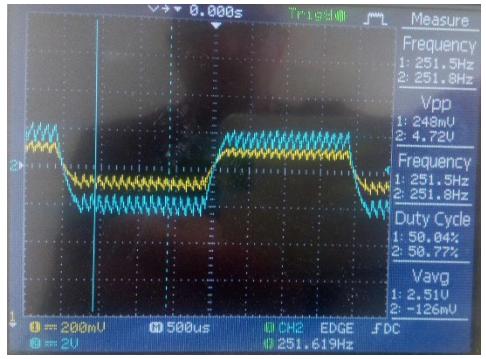
- ISR de fin de conversion (EOC) de ADCINT1
- GPIO32 indique les début/fin de l'ISR
- Lecture des résultats de conversion
- Acknoweldge interruption / clear Flag

```
interrupt void adca1_isr(void)
{
  GpioDataRegs.GPBSET.bit.GPIO32 = 1;   // Set
  ADC_Ia = AdccResultRegs.ADCRESULT0;
  ADC_Ib = AdcbResultRegs.ADCRESULT0;
  ADC_Ic = AdcaResultRegs.ADCRESULT0;

  ADC_Va = AdcaResultRegs.ADCRESULT1;
  ADC_Vb = AdccResultRegs.ADCRESULT1;
  ADC_Vc = AdcbResultRegs.ADCRESULT1;
  ADC_Vdc = AdcaResultRegs.ADCRESULT2;

  Ia = (ADC_Ia-ADC_IaOffset)*Igain;
  Ib = (ADC_Ib-ADC_IbOffset)*Igain;
  Ic = (ADC_Ic-ADC_IcOffset)*Igain;
  Vdc = (ADC_Vdc-ADC_VdcOffset)*Vgain;

  AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
…
  // Check if overflow has occurred
  if(1 == AdcaRegs.ADCINTOVF.bit.ADCINT1)
    {
    AdcaRegs.ADCINTOVFCLR.bit.ADCINT1 = 1; //clear INT1 overflow flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear INT1 flag
    }

 PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
  GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;   // Clear
}
```

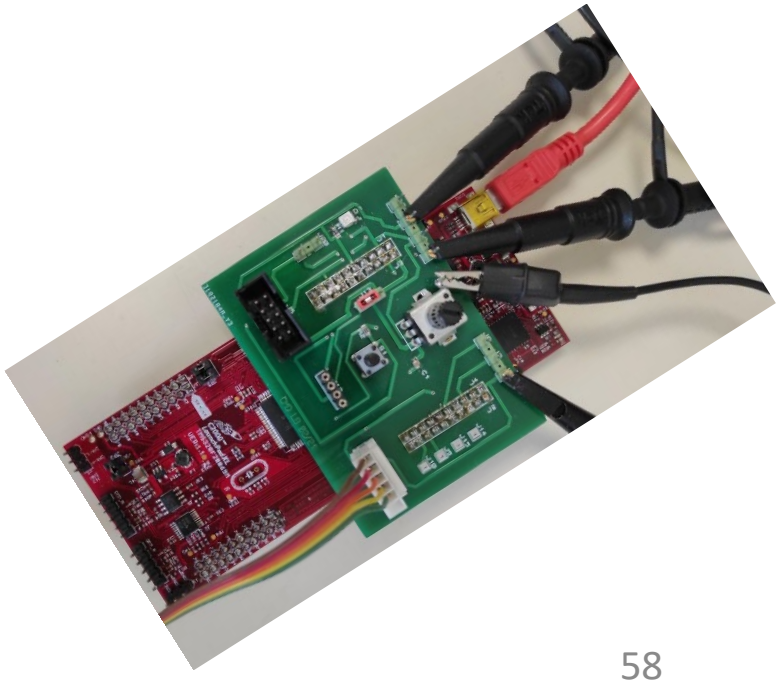
Target : Les applications possibles

- Contrôle des Convertisseurs Statiques (PWM, ADC, ISR, GPIO)
- Commande de moteurs triphasés
- Commande de drones, véhicules
- Commande des systèmes de contrôle (HVAC,…)

Merci pour votre attention



Discussions

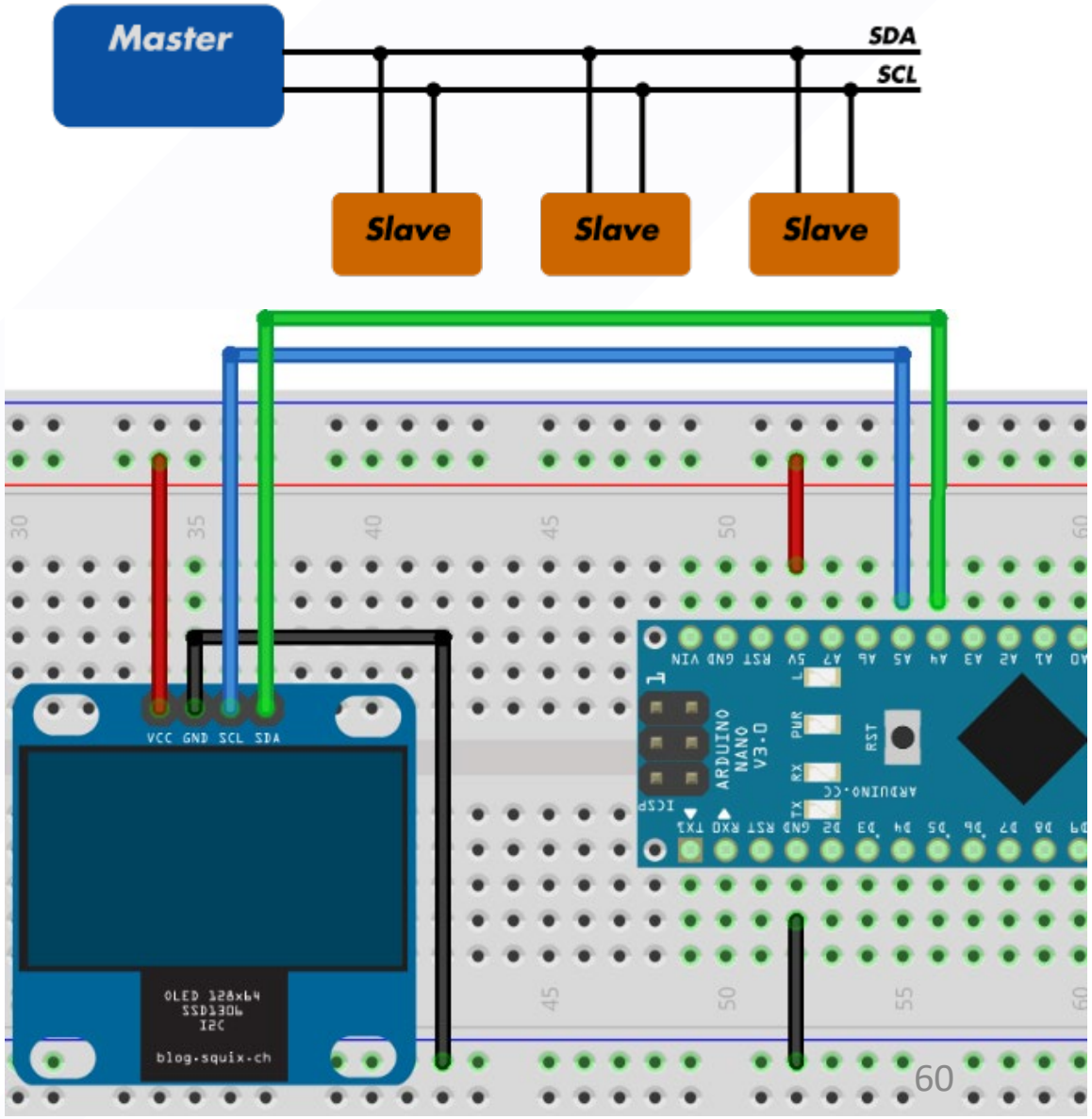


~~Peripherals~~

- Digital Input / Output Lines (I/O), pull-up, Change Notification (CN), wake
- Analogue to Digital Converter (ADC), SOC sources, Interrupt.
- Digital to Analogue Converter (DAC)
- Timer / Counter units, prescaler, cascadables, WatchDog
- Pulse Width Modulation (PWM), trigger ADC SOC
- Digital Capture Input Lines
- ~~Real Time Clock and Calendar (RTCC)~~
- Communication Interface Units
  - Serial Communication Interface (SCI) - UART
  - Serial Peripheral Interface (SPI)
  - Inter Integrated Circuit (I2C) – Bus
  - Controller Area Network (CAN)

Inter Integrated Circuit (I2C)

- Inter Integrated Circuits
- Synchronous Tx and Rx communication
- I2C bus: SCL (Clock) and SDA (Data)
- We do not bother with the communication protocol, headers, start, stop
- Each slave device has a specific address.
- For example:
  - The **Oled Screen** provided have default I2C address 0x78
  - The default I2C address for the **sensor BMP280** is 0x77 and the alternative I2C address is 0x76



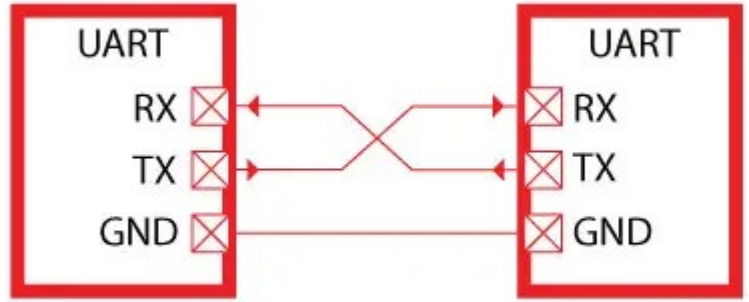
https://controllerstech.blogspot.com/2017/07/i2c-in-stm32.html

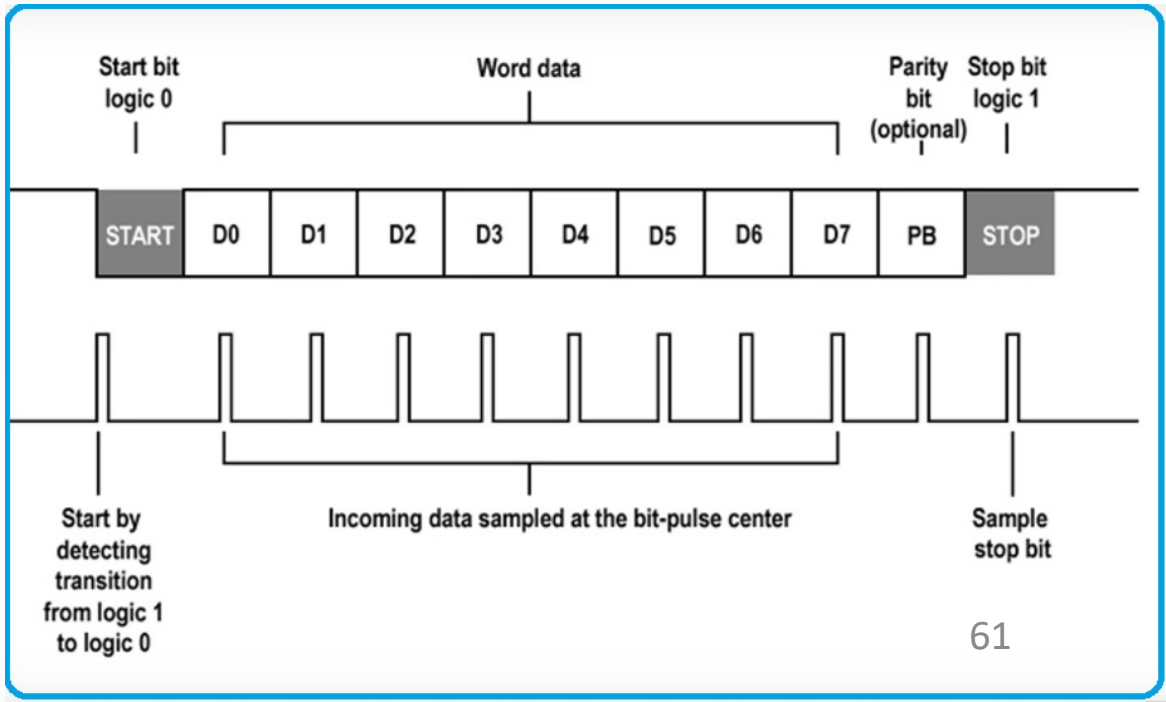
Universal Asynchronous Receiver Transmitter (UART)

- Serial communication
- Most standard communication protocol
- Exists at another voltage level (RS232) through transceivers
- Speed in bits per seconds (BAUDS). Common values are: 9600, 19200 and 115200
- The UART is not a fast transmission interface compared to SPI or I2C
- In Arduino, the syntax to use the Serial interface over the USB, to communicate with the PC is:

```
Serial.begin(115200);
```

- We can then use the other function to send or read data and text over the serial interface
- To write a text: `Serial.print("Hello World")`

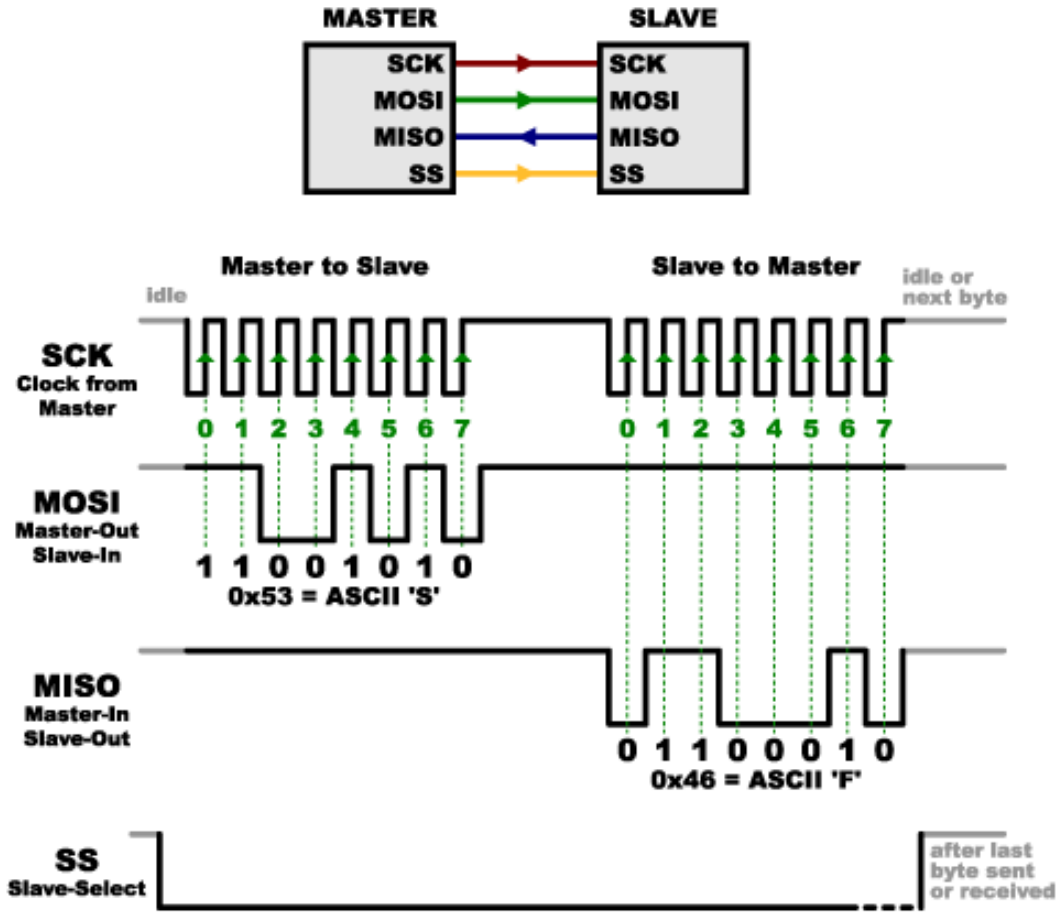




https://microcontrollerslab.com/uart-communication-working-applications/

Serial Peripheral Interface (SPI)

- Synchronous serial communication interface
- For short distances and high speed (Mbps)
- Generally it needs 4 lines between the master anc the slave
- The master imposes the clock and initiate the communication
  - SCLK: Serial Clock (output from master)
  - MOSI: Master Out Slave In (data output from master)
  - MISO: Master In Slave Out (data output from slave)
  - SS: Slave Select (often active low, output from master)
- More complex than I2C and UART (clock polarity and phase, Daisy-chained,…) but the fastest interface
- In Arduino, the syntax to use the SPI, to communicate with an external sensor is often hidden in the library of the sensor

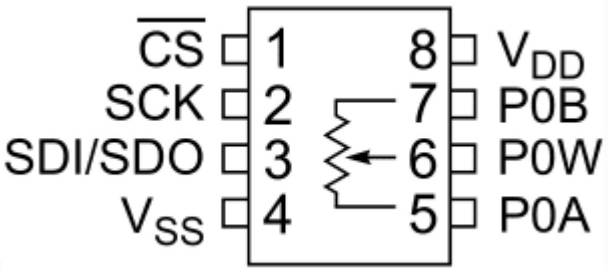


https://en.wikipedia.org/wiki/Serial_Peripheral_Interface
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi/all

Microcontrôleurs et périphériques

Serial Peripheral Interface (SPI)

- In Arduino, the syntax to use the SPI, to communicate with an external sensor is often hidden in the library of the sensor
- In this explicit example, we send 0x00 on the MOSI then an 8 bits between 0 and 128
- This number sets the output resistance at pin 6 of the MCP4131. A smaller number sets a lower resistance, and a larger number sets a higher resistance.



MCP4131
Digital potentiometer

```ino
#include <SPI.h>

void setup() {
    pinMode(10, OUTPUT); // set the SS pin as an output
    SPI.begin();         // initialize the SPI library
    Serial.begin(115200);
}

void loop() {
    digitalWrite(10, LOW);       // set the SS pin to LOW
    for(byte wiper_value = 0; wiper_value <= 128; wiper_value++) {
        SPI.transfer(0x00);     // send a write command to the MCP4131 to write at registry address 0x00
        SPI.transfer(wiper_value);    // send a new wiper value
        Serial.println(analogRead(A0)); // read the value from analog pin A0 and send it to serial
        delay(1000);
    }
    digitalWrite(10, HIGH);      // set the SS pin HIGH
}
```

https://www.circuitbasics.com/how-to-set-up-spi-communication-for-arduino/