CEI UPM | Centro de Electrónica Industrial

Digital Control of Power Converters: Piccolo Microcontroller Training Class I

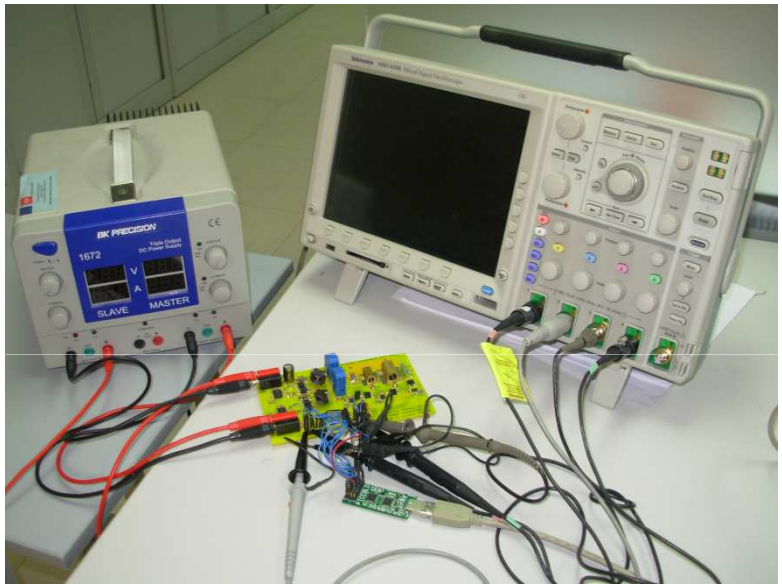
ceiupm.es

Universidad Politécnica de Madrid

POLITÉCNICA

MCU to implement the digital control of a switched power converter

- Specifically designed to control switched mode power converters: modules as ADC, PWM, comparators, timers…

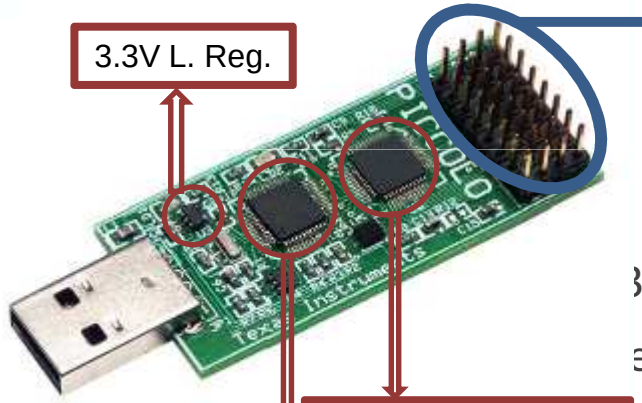


- Low cost digital controller
- No concurrency of calculations
- Interrupt based MCU
- Software: Code Composer Studio, based on Eclipse software framework, programmed in C/C++

Digital controller: Piccolo 28027

- TMS320F28027 Piccolo Microcontroller (Evaluation Board)
- Low Cost µC from Texas Instruments
- Device suitable to control Switch-Mode Power Supplies

- Mai
  - 32-bit f
  - esoluti
  - 8 PWMs (4 HR PWMs) ... Piccolo controller ... ighres
  - 2 Comparators
  - 3 Timers



Pinout

1 ADC-A7	2 ADC-A2 COMP1 (+VE)	3 ADC-A0 Vref-HI	4 3V3
5 ADC-A4 COMP2 (+VE)	6 ADC-B1	7 EPWM-4B GPIO-07	8 TZ1 GPIO-12
9 SCL GPIO-33	10 ADC-B6	11 EPWM-4A GPIO-06	12 ADC-A1
13 SDA GPIO-32	14 ADC-B7	15 EPWM-3B GPIO-05	16 5V0
17 EPWM-1A GPIO-00	18 ADC-B4 COMP2 (-VE)	19 EPWM-3A GPIO-04	20 SPISOMI GPIO-17
21 EPWM-1B GPIO-01	22 ADC-B3	23 EPWM-2B GPIO-03	24 SPISIMO GPIO-16
25 SPISTE GPIO-19	26 ADC-B2 COMP1 (-VE)	27 EPWM-2A GPIO-02	28 GND
29 SPICLK GPIO-18	30 GPIO-34 (LED)	31 PWM1A-DAC (Filtered)	32 GND

Digital controller: Piccolo 28069

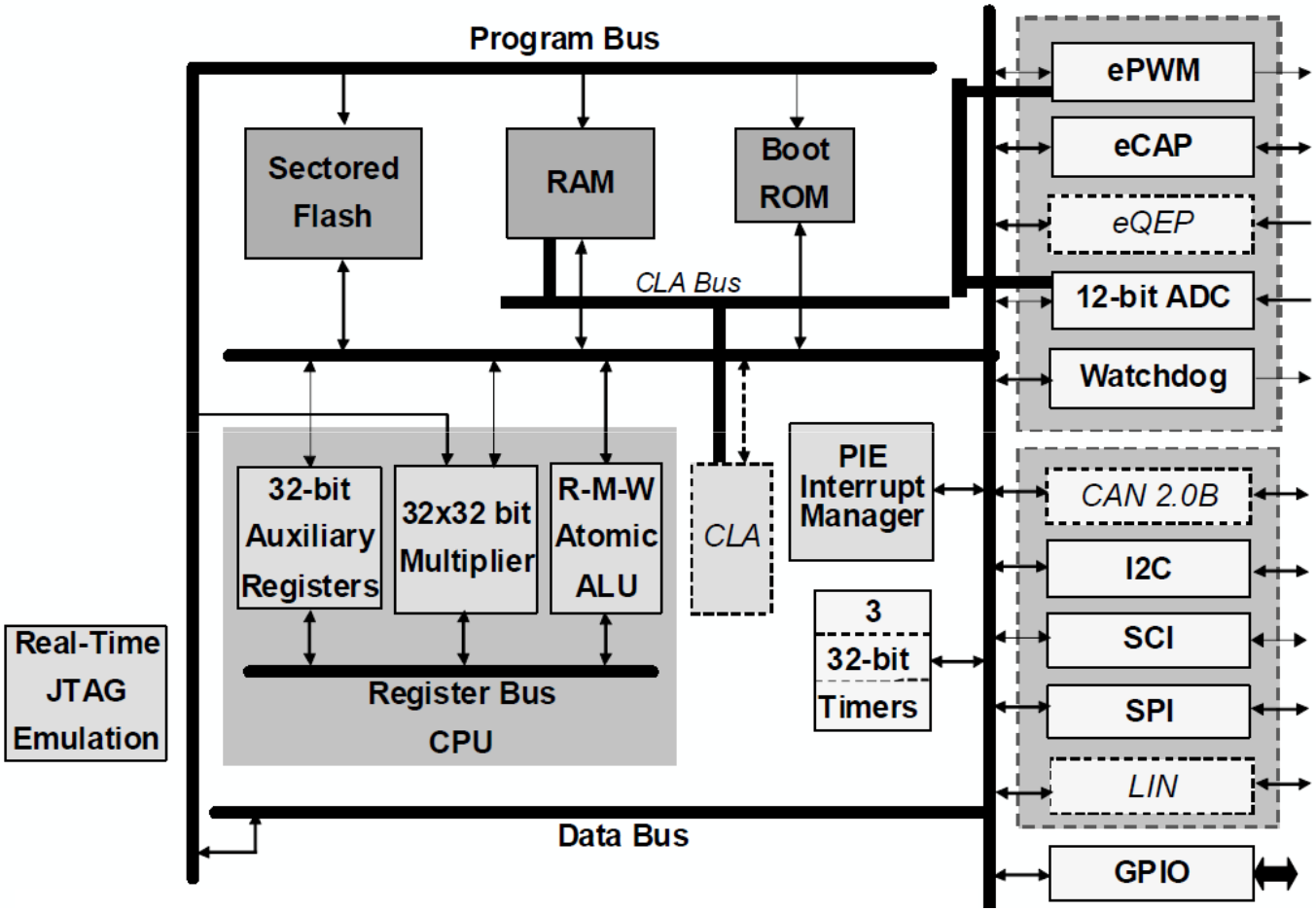
- TMS320F28069 Piccolo Microcontroller (Evaluation Board)
- Low Cost µC from Texas Instruments
- Device suitable to control Switch-Mode Power Supplies

- Mair
  -
  -
  -
  -
  -
  -

1 ADC-A6 COMP3(+VE)	2 ADC-A2 COMP1 (+VE)	3 ADC-A0	4 3V3
5 ADC-A4 COMP2 (+VE)	6 ADC-B1	7 EPWM-4B GPIO-07	8 TZ1 GPIO-12
9 SCLA GPIO-33	10 ADC-B6 COMP3(-VE)	11 EPWM-4A GPIO-06	12 ADC-A1
13 SDAA GPIO-32	14 ADC-B0	15 EPWM-3B GPIO-05	16 5V0 (Disabled by Default)
17 EPWM-1A GPIO-00	18 ADC-B4 COMP2 (-VE)	19 EPWM-3A GPIO-04	20 SPISOMIA GPIO-17
21 EPWM-1B GPIO-01	22 ADC-A5	23 EPWM-2B GPIO-03	24 SPISIMOA GPIO-16
25 SPISTEA GPIO-19	26 ADC-B2 COMP1 (-VE)	27 EPWM-2A GPIO-02	28 GND
29 SPICLKA GPIO-18	30 GPIO-34 (LED)	31 PWM1A-DAC (Filtered)	32 GND

on ePWM

MCU Architecture



What can be done with Piccolo DSC…device characteristics

- Interrupt based device: Uses interrupts to synchronize the tasks to the proper event: Up to 96 different interrupt sources

- Signals measurement –> Peripheral 12-bit ADC module (up to 16 inputs). The resolution of the ADC is 0V-3.3V

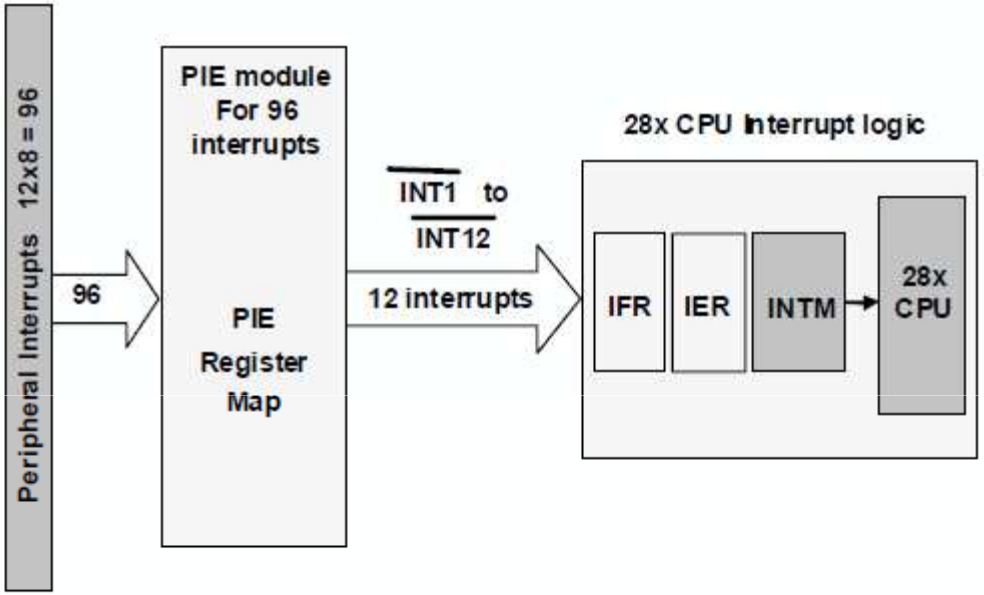
- Generate PWM control signals: 4 modules (each one with 2 PWM signals)

- General Purpose Input Outputs (GPIO): Up to 22 GPIO signals divided in two ports

- Internal comparators and timers: Up to two comparators and three 32-bit CPU timers

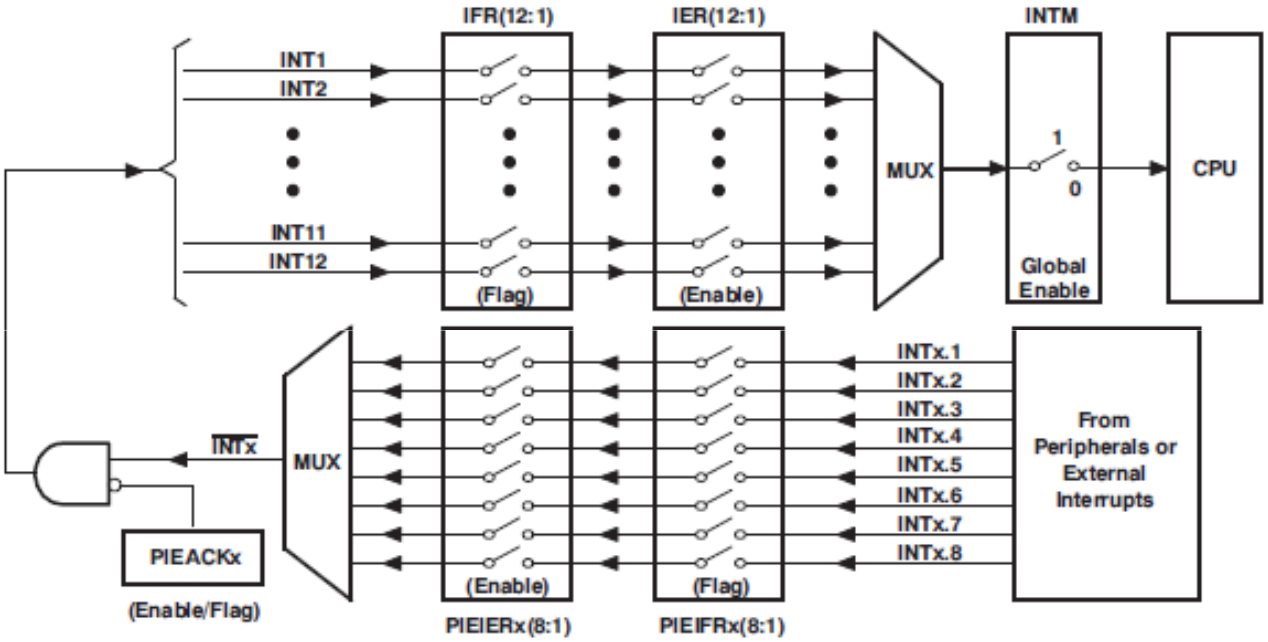
System Control and Interrupts

- 96 dedicated PIE vectors
- No software decision making required
- Direct access to RAM vectors
- Auto flags update
- Concurrent auto context save



System Control and Interrupts II

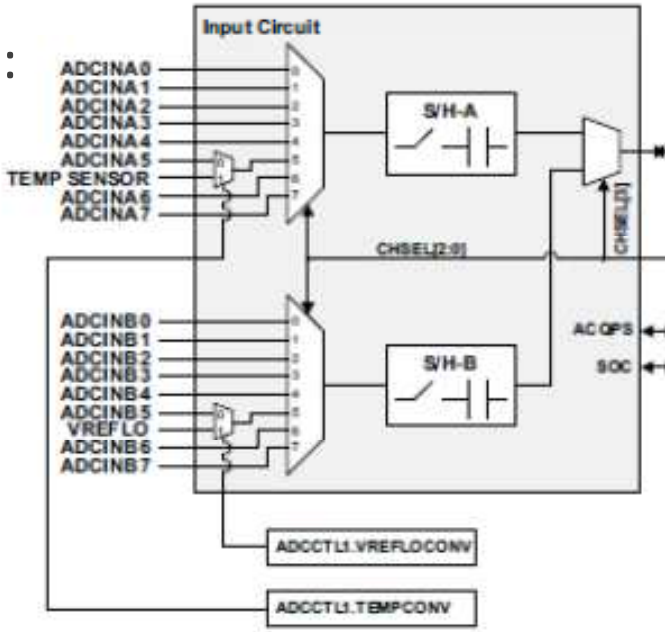
- It operates by **interrupts**: Up to 96 different sources



- Habilitations: Interrupts have to be enabled in three registers:
  - PIE level (PIEIERx) → CPU level (IER) → Global interrupt (INTM)

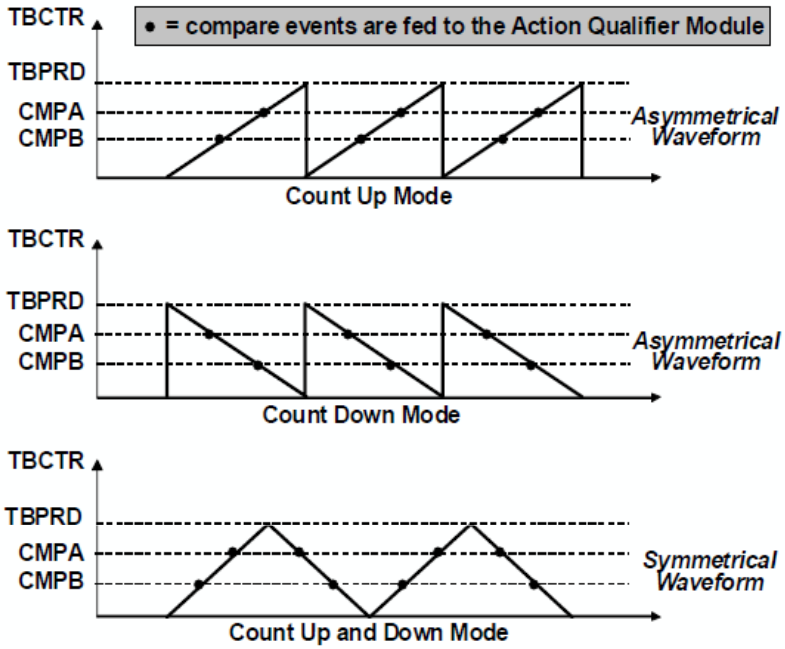
ADC Peripheral Module

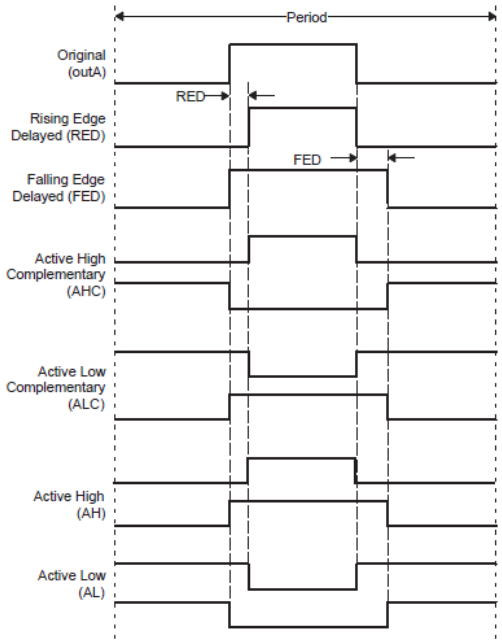
- Signals measurement –> Peripheral **ADC** module
  - Full scale 0V-3.3V (12-bits=> 0-4095)
  - 16 multiplexed inputs and 16 result registers to store conversion results
  - Multiple trigger sources: ePWM, GPIO XINT2, 3 CPU timers, ADCINT1/2
  - Different measurement strategies:
    - Round robin or managing priorities
    - Simultaneous measurements

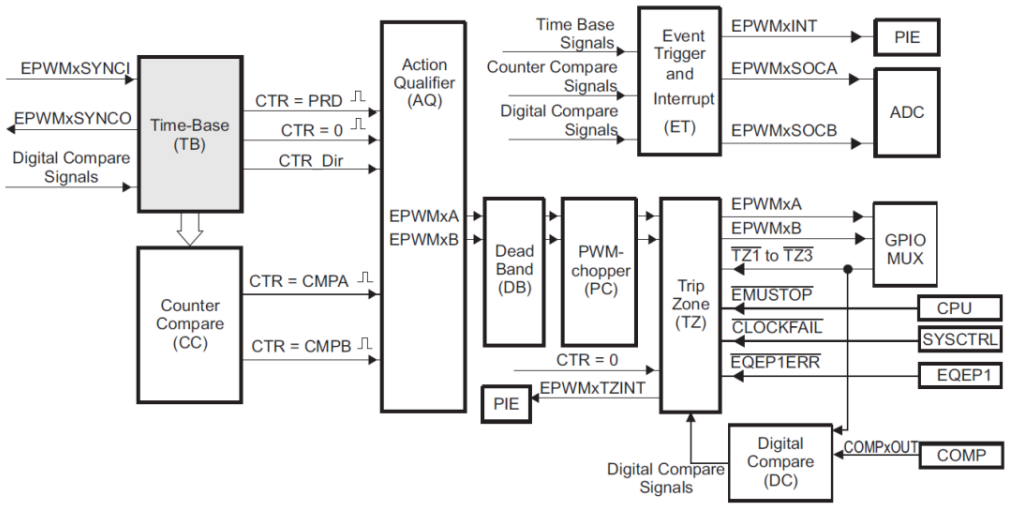


ePWM Peripheral Module

- Three basic PWM modulation modes:
  - Asymmetrical (count up or down)
  - Symmetrical



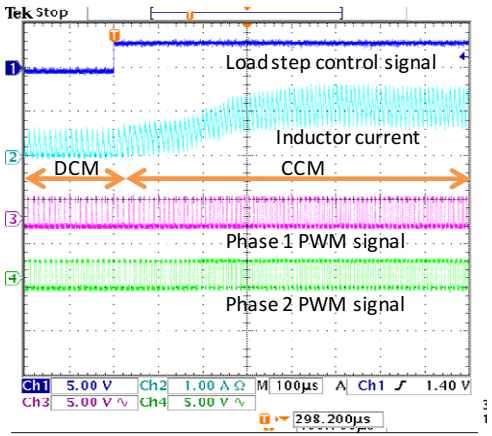




- Several dead-band time mode

What can be done with Piccolo DSC…in the system to be controlled

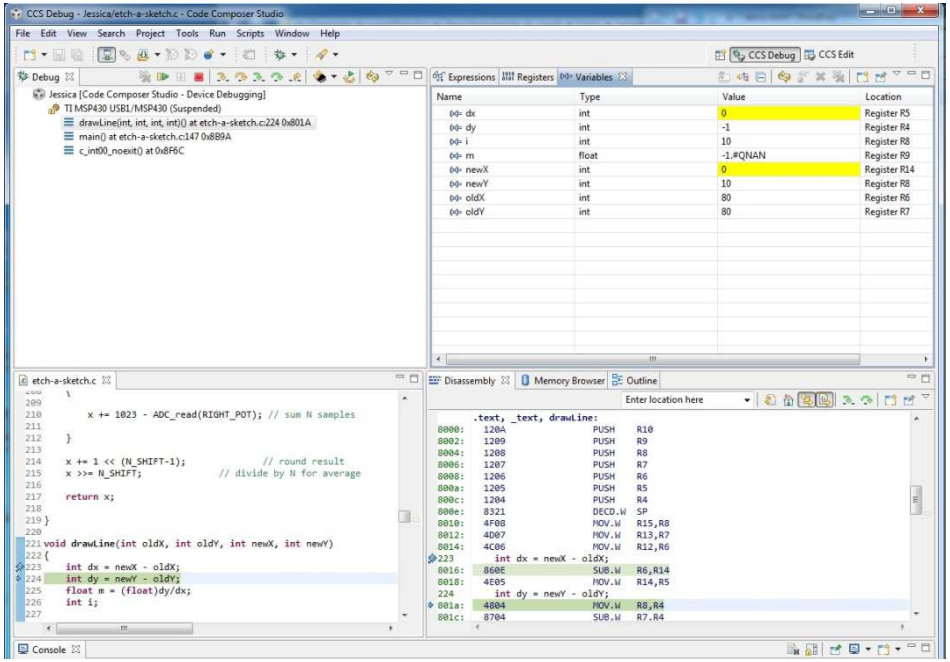
- Control up to four pairs of power switches
- Generate a phase delay between the ePWM control signals
- Dead-band time between control signals
- Variable frequency control
- Implementation of regulators in the discrete domain
- Synchronization of the measurements tasks
- Protections implementation
- Soft-Start transient



- Easy implementation of more complex functionalities:
  - Dynamic voltage scaling (DVS), phase switching as a function of the load, switching between different conduction modes, minimum time control…

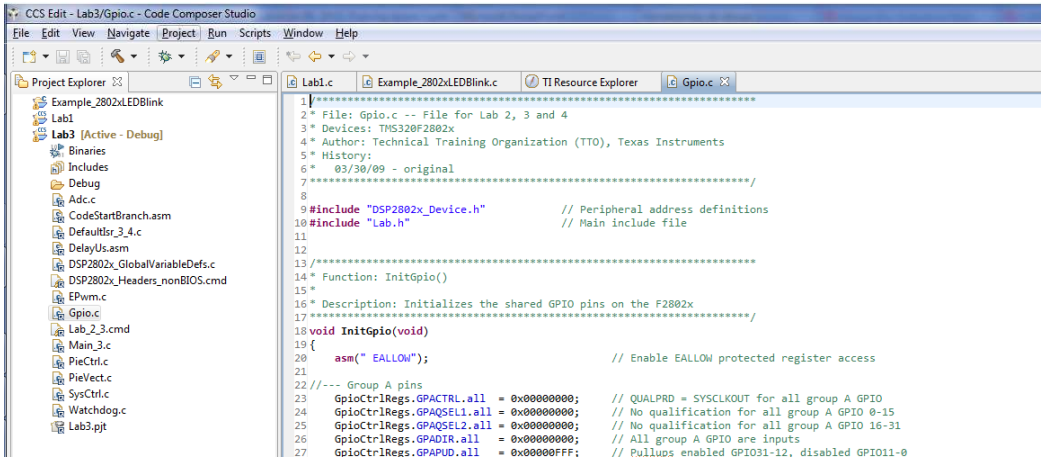
Code Composer Studio Installation

- Code Composer Studio v5:
  - Already installed in the computers
  - To install in other computer, available from shared folder SERVIDORDISCOS\MASTER EI\Digital Control
  - Select XDS100 Emulator option (free)



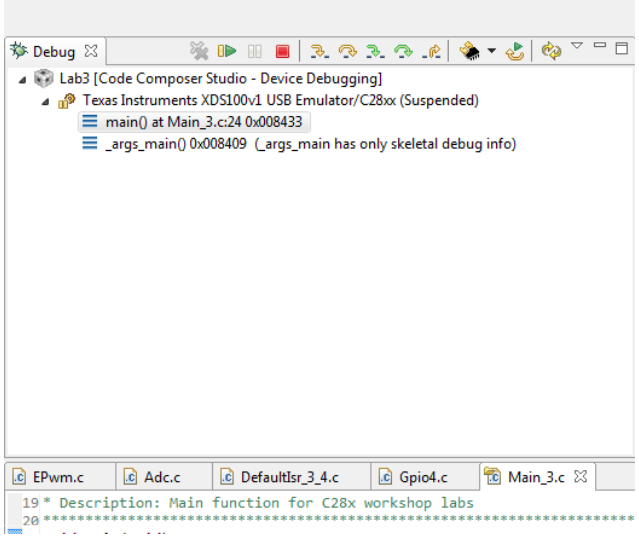
Design Flow : Configuring Piccolo

- Main part of the code is dedicated to set properly the registers:
  - Define the interrupts and the synchronization if necessary
  - General Purpose Input and Outputs (GPIO)
  - Number and type of ePWM
  - Number of ADC inputs
  - If necessary, set timers and comparators



- <u>Modularity of the code and Real-time mode</u>: Very useful to debug and to watch the values of the parameters of the system

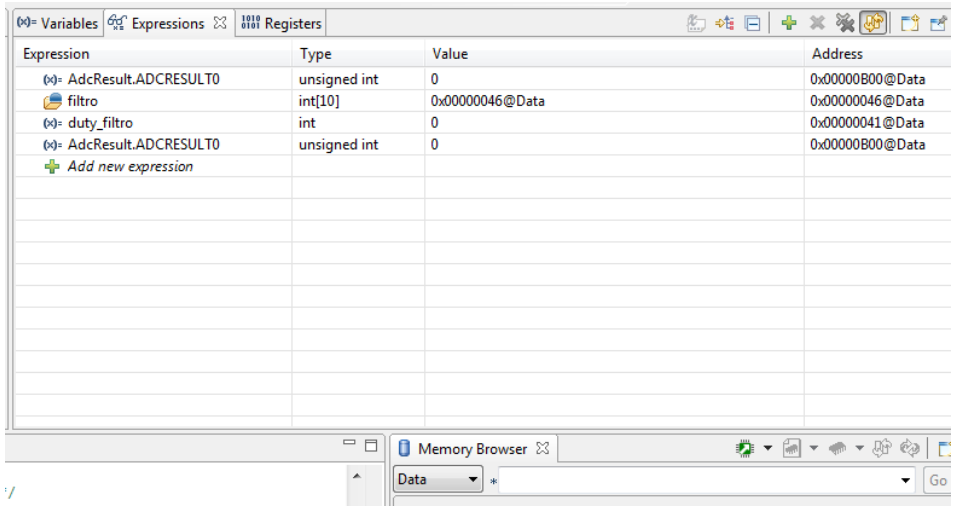
SW to control Piccolo µC: Code Composer Studio



- Code Composer Studio Characteristics:
  - Modular structure
  - Integrates: Edit, code generation and debug
  - Fast Interrupt Manager
  - Watch window + Real time mode

A **.pjt** proyect invokes all necessary tools to build the proyect (compiler, assembler, linker).

It also creates **.out** file (executable for the µC) **.map** file (memory usage and section addresses)



Provided material

- Texas Instruments Piccolo one day tutorial (4 Labs)
  - Includes C28x1DayWorkshop file
- Piccolo main datasheets:
  - Piccolo Microcontrollers 2802X: TMS320F28027
  - Enhanced Pulse Width Modulator: Piccolo ePWM
  - Analog to Digital Converter: Piccolo ADC
  - Code Composer Studio Guide
  - Piccolo System Control and interrupts
- Documentation folder:
  - Additional documentation

Basic Functionalities

- Open **Lab1.pjt** file from CCS
- Objectives:
  - Use of Code Composer Studio
  - Basic Piccolo controller operation
  - Debug/Build Options
  - Different ways to access memory information
  - Watch Window to check/modify local & global variables in real time

Control Peripherals

- Open **Lab3.pjt** file in Code Composer Studio
- Objectives:
  - Understand basic issues of interrupts, ADC and ePWM

- Description:
  - ePWM1A is used to generate a 2kHz PWM waveform
  - ePWM2 (50kHz)is triggering the ADC on period match using SOC A trigger→ADC conversion is set at a 50 kHz sampling rate
  - Program performs conversion on ADC channel A0 (ADC-A0 –pin 3 Ev B.)
  - Data is continuously stored in the buffer
  - Data is displayed using the graphing feature of CCS

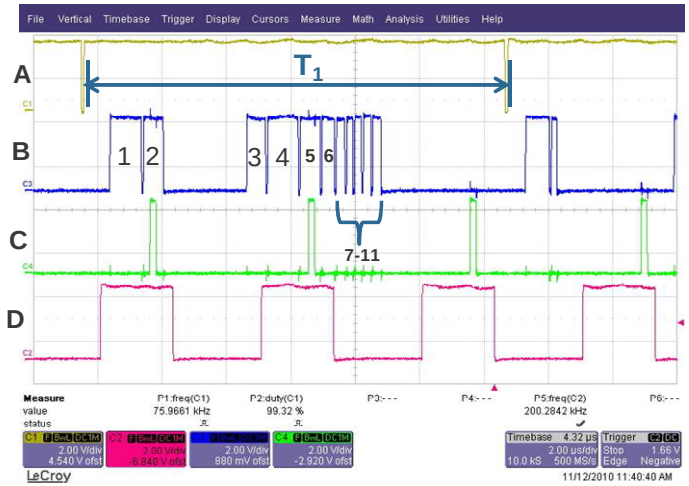
Exercise 1

- Using **Lab3** project, calculate the mean value of a PWM signal (filtering a PWM)
- Practice:
  - ADC
  - interrupts
  - ePWM
- Tasks:
  - Modify ePWM1A frequency and adjust the number of points to be acquired. Example: create an array of 10 points for frequencies of 50kHz (ePWM2) and 5kHz (or 10kHz) (ePWM1)
  - <u>Write the code of the filter in the project file “DefaultIsr_3_4.c”:</u>
    - Measure X times in a period of ePWM1 and store in an array o x values
    - Obtain the “duty cycle” obtaining the mean of the measured values once the array is written with the new period values

Exercise 2

- Obtain the computation time of a task and the location in the period using an auxiliary signal:
  - Enable a GPIO signal
  - Set GPIO to high-state, then create an operation with integer data and finally set again to low-state.
  - Check with the oscilloscope the length of the task
  - Now define floating point data and do the same operation
  - Check the length in the oscilloscope the length of the task

- To enable a GPIO signal:
  - In GPIO.c, set the GPAMUX (or GPBMUX) properly
  - Configure the GPIOx as an output



Exercise 3

- Create a pulse of 500ms width on GPIO34 – Output can be seen also in the LED2 of the board (ON for low level of GPIO34)
- Practice:
  - GPIO
  - interrupts

- Tasks:
  - In the file “DefaultIsr_3_4.c” add the code to control the GPIO34
    - Create a “GPIO34_count” counter
    - Calculate the number of interrupts to achieve
    - Toggle (change the value) of GPIO34 signal
      - GpioDataRegs.GPBTOGGLE.bit.GPIO34=1

Exercise 4

- Use of ePWM
  - 1.- Configure the ePWM2 to obtain the control pulses
  - 2.-Synchronize two PWM modules (useful for obtain more accuracy in the exercise 1):
    - Synchronization of modules ePWM1 and ePWM2
    - Configure ePWM2 with a 180° phase delay to ePWM1
    - Use of Time-Base module registers:
      - TBCTL.SYNCOSEL
      - TBCTL.PHSEN
    - Information about the values in the ePWM module datasheet

Exercise 5

- Use of Dead-band time: Generate complementary signals with dead-band time
- Practice:
  - GPIO: Habilitation of GPIO
  - Interrupts: Use of different sources to enter interrupt and to measure different signals

- It is also possible to generate the complementary signal by using the registers of the PWM (without using Dead-band time control)

CEI UPM | Centro de Electrónica Industrial

Digital Control of Power Converters: Piccolo Microcontroller Training Class II

ceiupm.es

Universidad Politécnica de Madrid

POLITÉCNICA

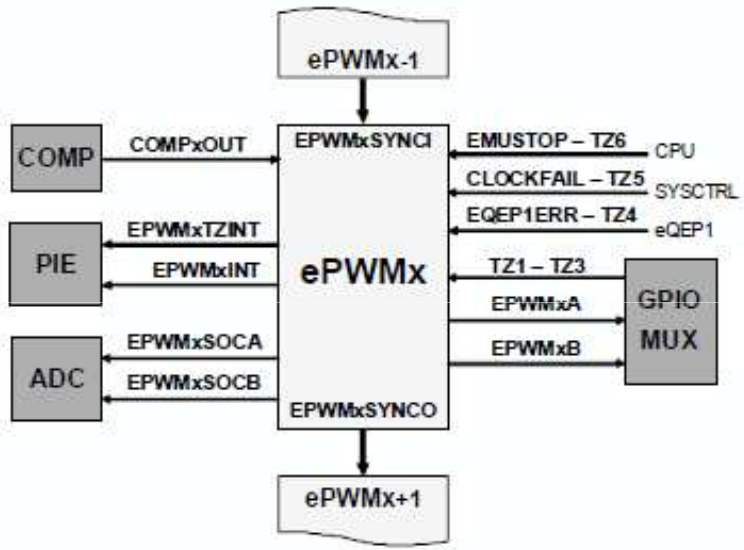
Schedule of Class II

- Training Class II:
  - Peripheral modules of the Piccolo DSC (ePWM, ADC)
  - Continuation of the class I exercises:
    - Exercise 1: Calculate the mean value of the PWM signal (duty cycle)
    - Exercise 2: Obtain the calculation time of a task for different types of data
    - Exercise 3: Obtain a 500ms-period pulsed signal
    - Exercise 4: Obtain the waveforms of ePWM1A and ePWM2A signals
    - Exercise 5: Obtain complementary PWM signals using Dead-band submodule
    - Exercise 6: Synchronize ePWM1 and ePWM2 modules and configure a phase delay

ePWM Peripheral Module

- **Connections:**

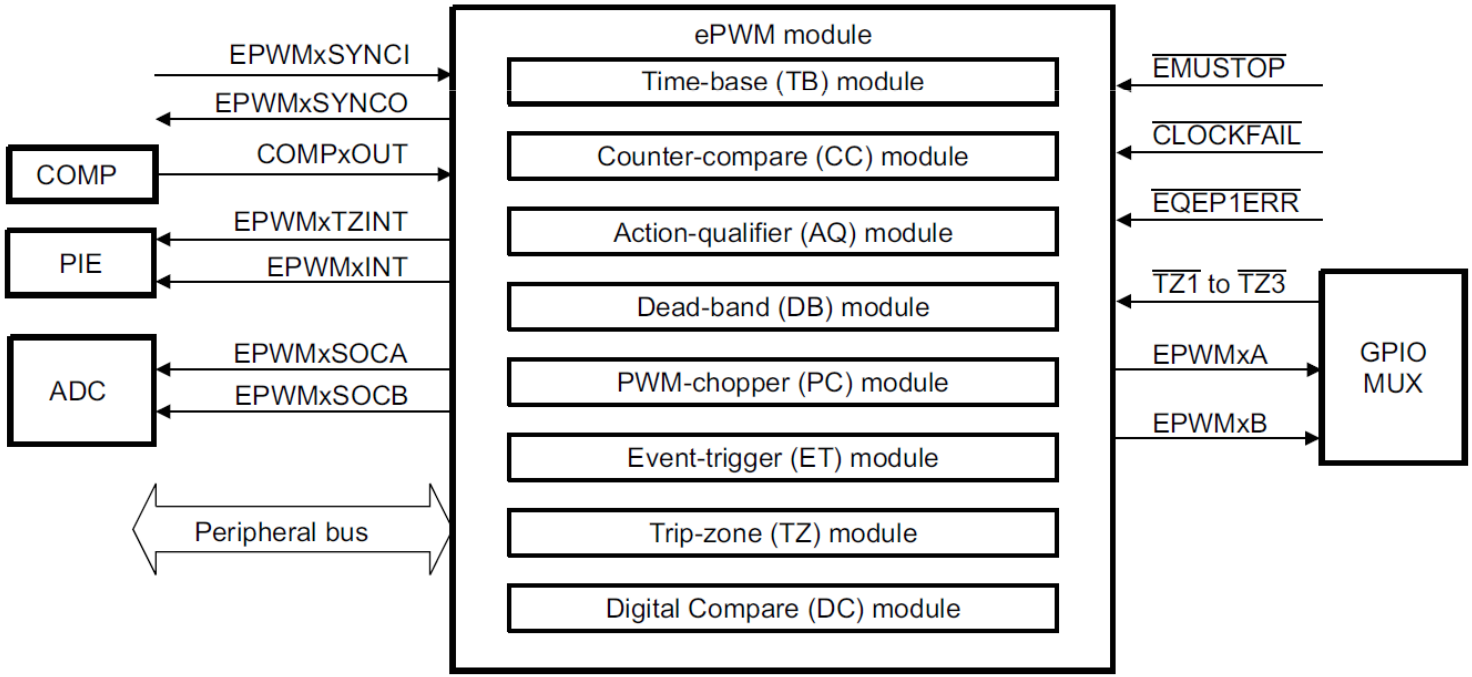
  - **Inputs**
    - Synchronization ePWMx-1
    - Comparator output
    - Trip Zone TZx signals



  - **Outputs**
    - EPWMxA and EPWMxB signals
    - Interrupts to PIE
    - EPWMxSOCA and B to ADC
    - Synchronization signal to next module (ePWMx+1)

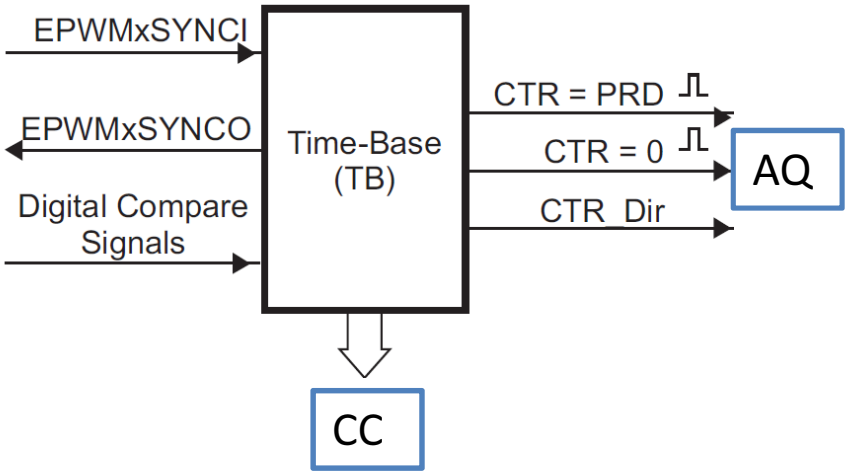
ePWM Peripheral Module

- Submodules:
  - Time-base (TB)
  - Counter-compare (CC)
  - Action-qualifier (AQ)
  - Dead-band (DB)
  - Trip-zone (TZ)
  - Event-trigger (ET)
  - PWM-chopper (PC)
  - Digital Compare (DC)



ePWM Time-base (TB) submodule

- Main functions
  - Configure period
  - Define the count mode
  - Set the Time-Phase
  - Synchronize the TB counter
  - Set the source of the Synchronizatic
    - Input signal
    - TBCTR =0 (=CMPB)
    - No sync signal
  - Counter compare
    - Generate events at the switching events



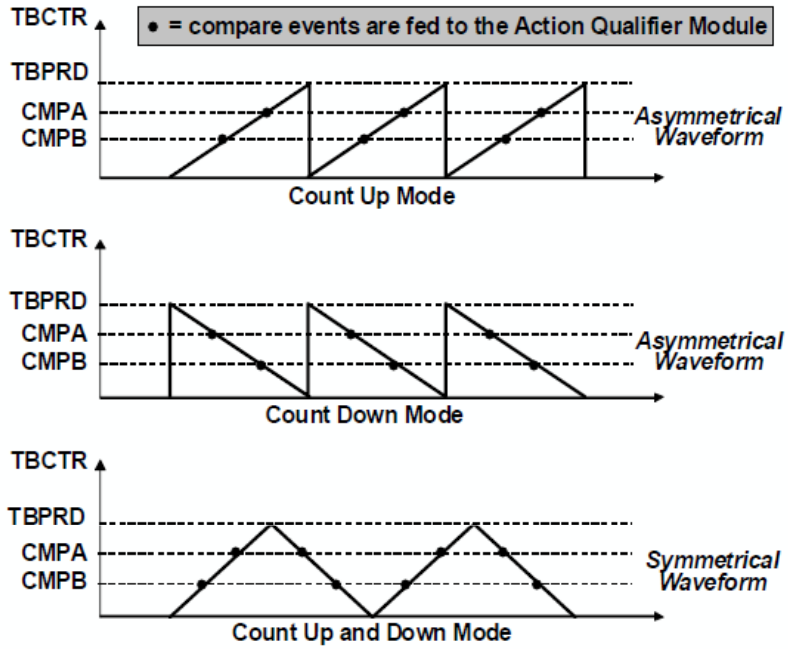
Main Registers

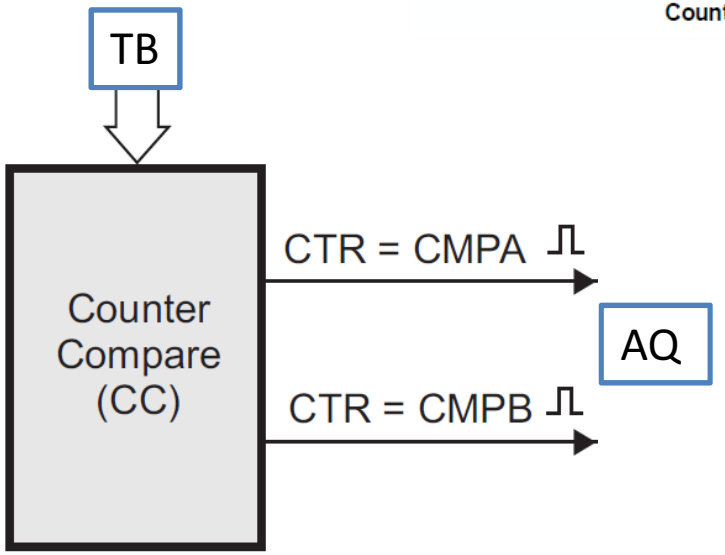
Time Base Control Register: TBCTL
Time base Period: TBPRD
Time Base Phase: TBPHS
Time base Counter: TBCTR

Defined as the number of clock periods

ePWM Counter-compare (CC) submodule

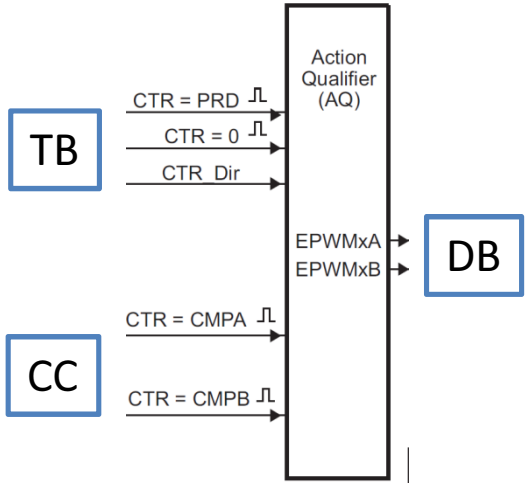
- Three basic PWM modulation modes:
  - Asymmetrical (count up or down)
  - Symmetrical
- Each ePWM allows two comparison events (CMPA and CMPB)
- Additional external sources of duty cycle can be defined



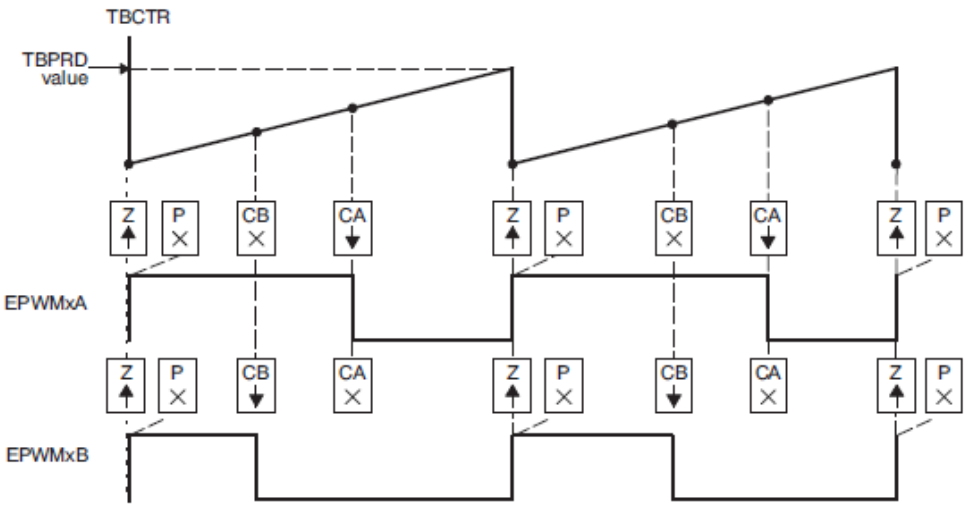


ePWM Action-qualifier (AQ) submodule

- Defines the actions when an event occurs
  - EPWMA or B pushed:
    - High
    - Low
    - Toggle



Example 1: ePWM1 configured up-down, single edge asymmetric independent modulation A and B



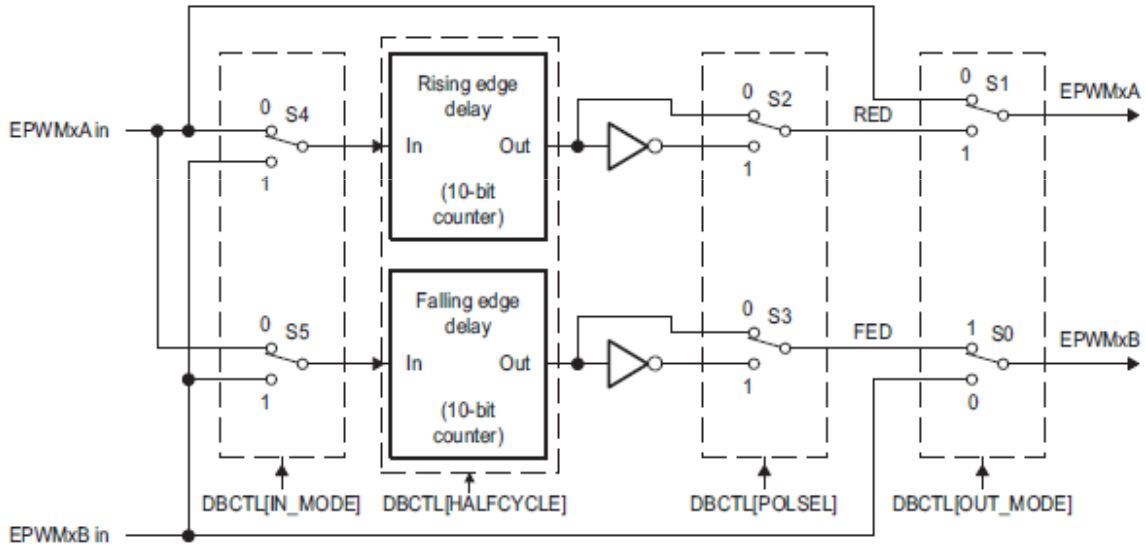
EPwm1Regs.AQCTLA.bit.ZRO
EPwm1Regs.AQCTLA.bit.CAD
EPwm1Regs.AQCTLB.bit.ZRO
EPwm1Regs.AQCTLB.bit.CBD

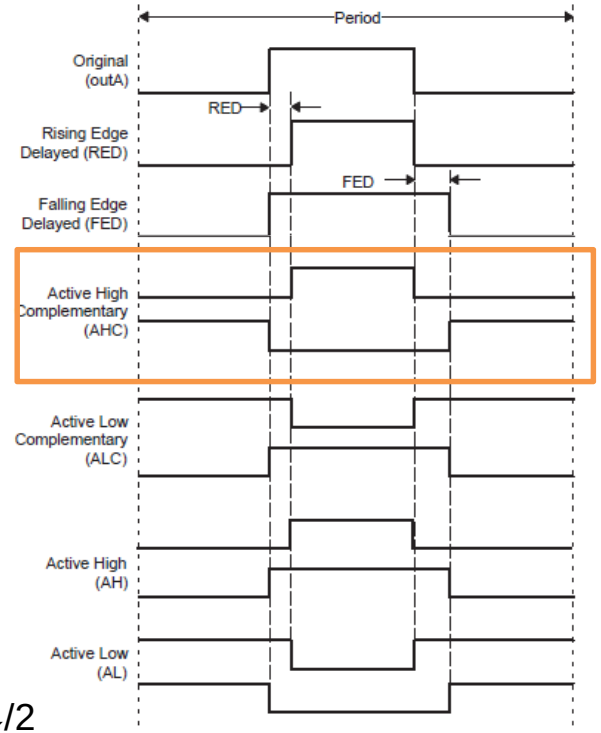
ePWM Dead-band (DB) submodule

- Possibility to implement several dead-band time modes
- Rising Edge Delay (RED)
- Falling Edge Delay (FED)

$FED = DBFED \times T_{TBCLK}/2$

$RED = DBRED \times T_{TBCLK}/2$





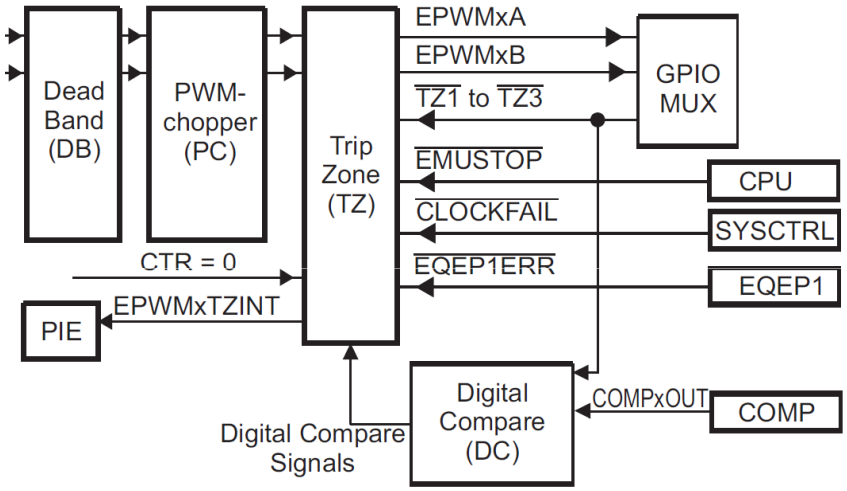
S0 & S1: DBCTL[OUT_MODE]
S2 & S3: DBCTL[POLSEL]
S4 & S5: DBCTL[IN_MODE]

$FED = DBFED \times T_{TBCLK}/2$

$RED = DBRED \times T_{TBCLK}/2$

ePWM Trip-Zone (TZ) submodule

- Each ePWM module is connected to six TZx signals (active low)
- Those signals indicate an external fault or trip condition
- ePWM outputs can be programmed to respond to them changing the output to:
  - High or low state
  - High-impedance
  - No action



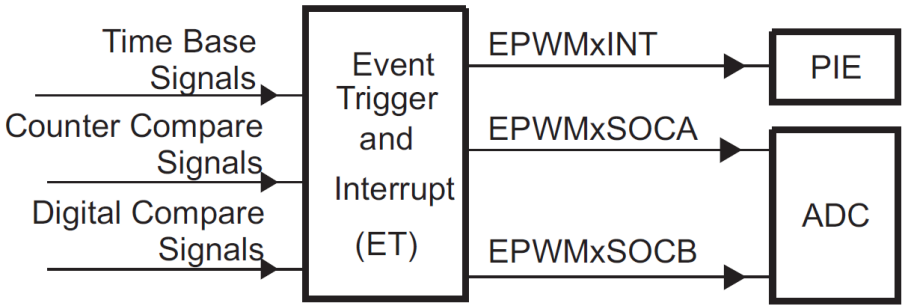
- TZ signals can be configured to be:
  - Cycle-by-cycle tripping → For **current limiting**
  - One shot trip→ For major **short-circuit** or **over-current**
- Allows sw-forced tripping

Trip-Zone Registers

TZSEL[OSHT1] : Enables TZ1 signal to be a one-shot event source
TSEL[CBC3] : Enables TZ3 signal to be a CBC source
TZCTL[TZA] : Defines the action of EPWMxA
TZCTL[TZB] : Defines the action of EPWMxB

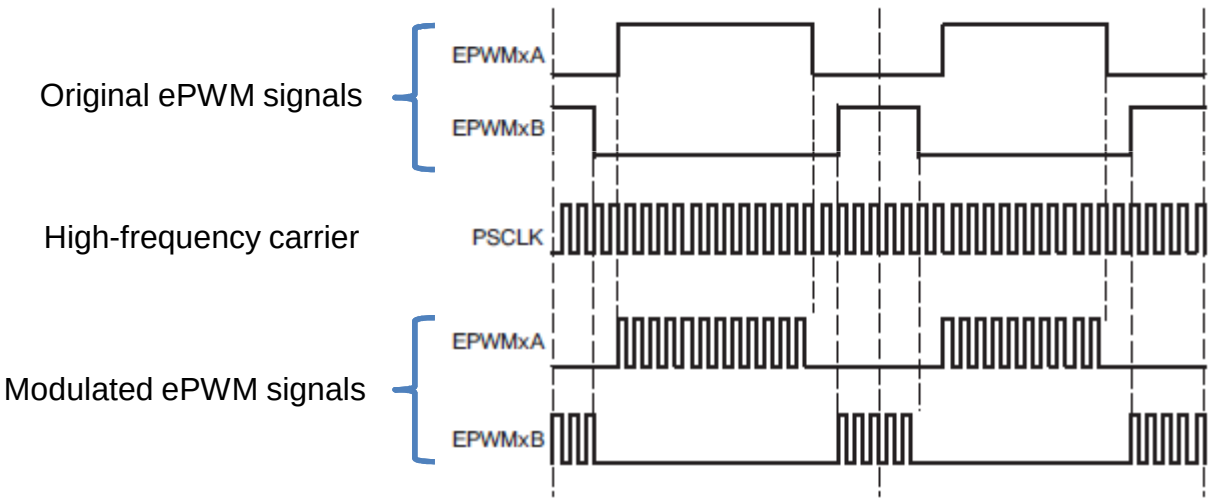
ePWM Event-Trigger (ET) submodule

- Receives event inputs generated by Time-base, Counter-compare, and Digital Compare submodules
- Interrupts the CPU and/or the ADC SOC
- Allows software forcing
- Each PWM module has one PIE interrupt and two SOC signals connected to the ADC (ePWMxSOCA and ePWMxSOCB)
- Prescaling logic:
  - Issues the interrupt requests and ADC SOC at (every/second/third event)



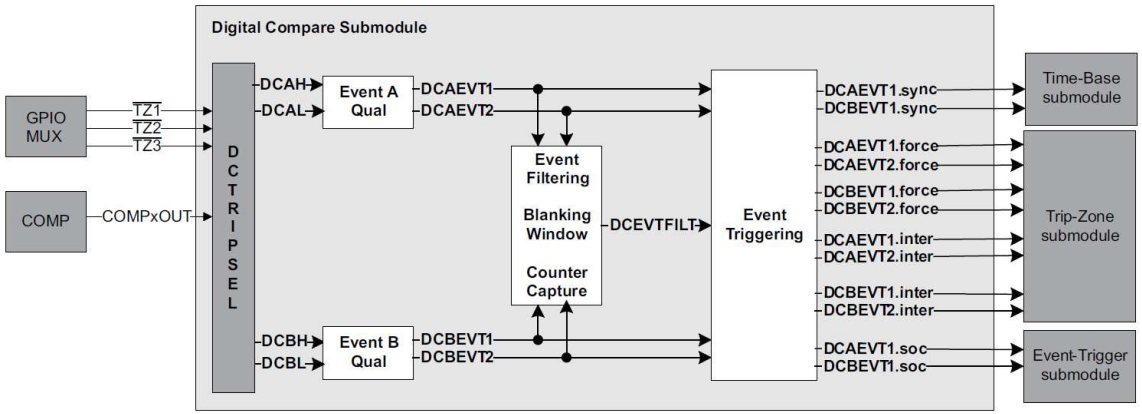
ePWM PWM-Chopper (PC) submodule

- Allows the modulation of a high-frequency carrier generated by the action-qualifier and dead-band submodules



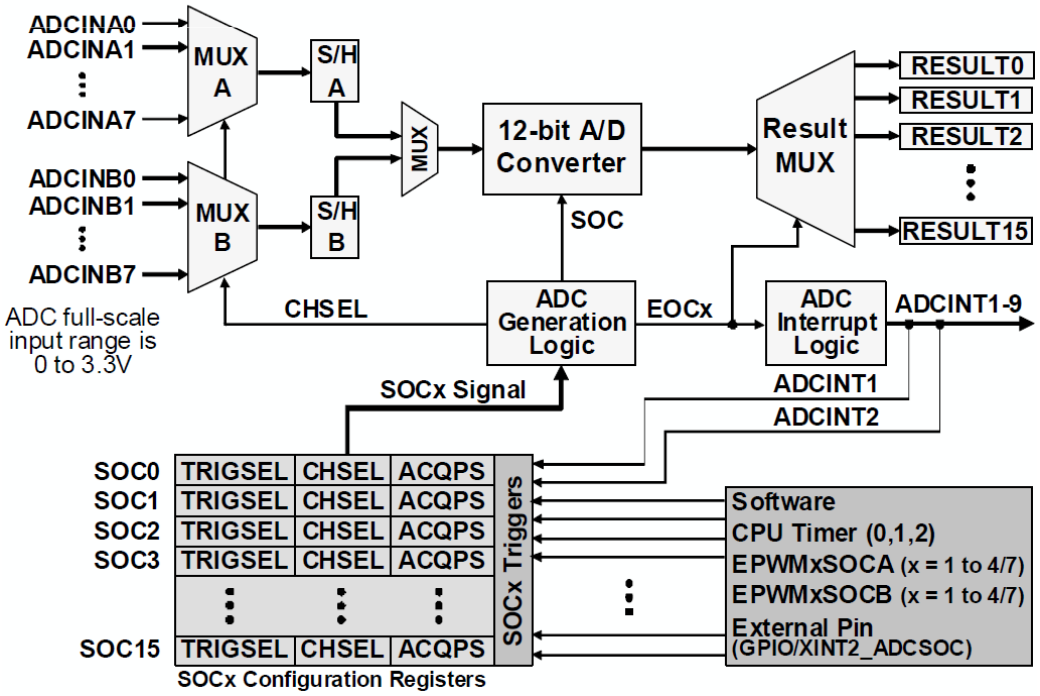
ePWM Digital Comparator (DC) submodule

- Compares external signals to the PWM module to generate directly events (or filter events) to:
  - Time-base submodule
  - Trip-zone submodule
  - Event-trigger submodule
- COMPxOUT, TZ1, TZ2 & TZ3 generate DCAH, DCAL, DCBH, DCBL
- Those signals can generate:
  - Trip-zone interrupt
  - ADC SOC
  - Force an event
  - Synchronization event to the TBCTR module



ADC Peripheral Module

- Signals measurement –> Peripheral **ADC** module
  - Full scale 0V-3.3V (12-bits=> 0 to 4095)
  - 16 multiplexed inputs and 16 result registers to store conversion results (ADCRESULTx)
  - Multiple trigger sources: ePWM, GPIO XINT2, 3 CPU timers, ADCINT1/2



ADC Peripheral Module

- The ADC is SOC based: each SOC defines the single conversion of a single channel
- Three main fields to configure SOC (each one has independent conf.):
  - Trigger source (TRIGSEL)
  - Acquisition window size (ACQPS)
  - Channel to convert (CHSEL)
- Example: ADCSOCxCTL is composed of three registers

15 – 11	10	9 – 6	5 – 0
TRIGSEL	Reserved	CHSEL	ACQPS
R/W-0	R-0	R/W-0	R/W-0

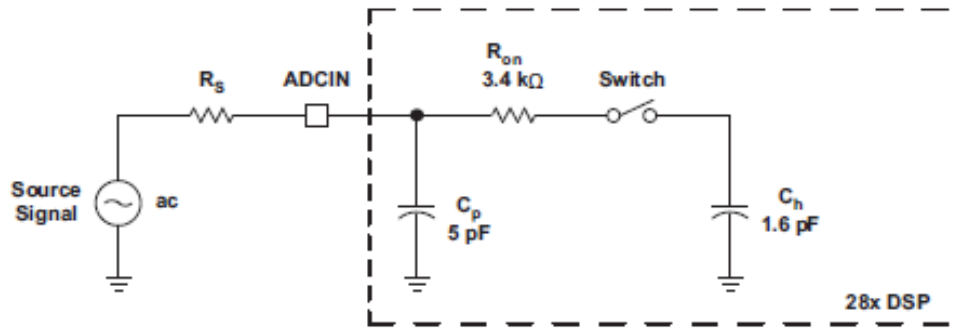
- Acquisition window size:
  - Minimum time is 6 cycles (+1)=7 to add to 13 conversion time cycles

ADC Clock	ACQPS	Sample Window	Conversion Time (13 cycles)	Total Time to Process Analog Voltage[1]
60MHz	6	116.67ns	216.67ns	333.34ns
60MHz	8	150.00ns	216.67ns	366.67ns
60MHz	10	183.33ns	216.67ns	400.00ns
60MHz	14	250.00ns	216.67ns	466.67ns
60MHz	25	433.33ns	216.67ns	650.00ns
40MHz	6	175	325ns	500.00ns
40MHz	25	625	325ns	950.00ns

(1) The total times are for a single conversion and do not include pipelining effects that increase the average speed over time.

ADC Peripheral Module

- ADCINx Input Model



- Trigger:
  - Software
  - CPU Timers 0/1/2
  - ePWMx SOCA and SOCB
  - External XINT2 SOC
  - ADCINT1 & ADCINT2 can be fed back to start another conversion

ADC Peripheral Module

- Channel Selection
  - Each SOC can convert any of the available ADCIN inputs

- ADC Conversion Priority:
  - Round-robin by default → From SOC0 to SOC15
  - It can be configured the number of high priority SOC

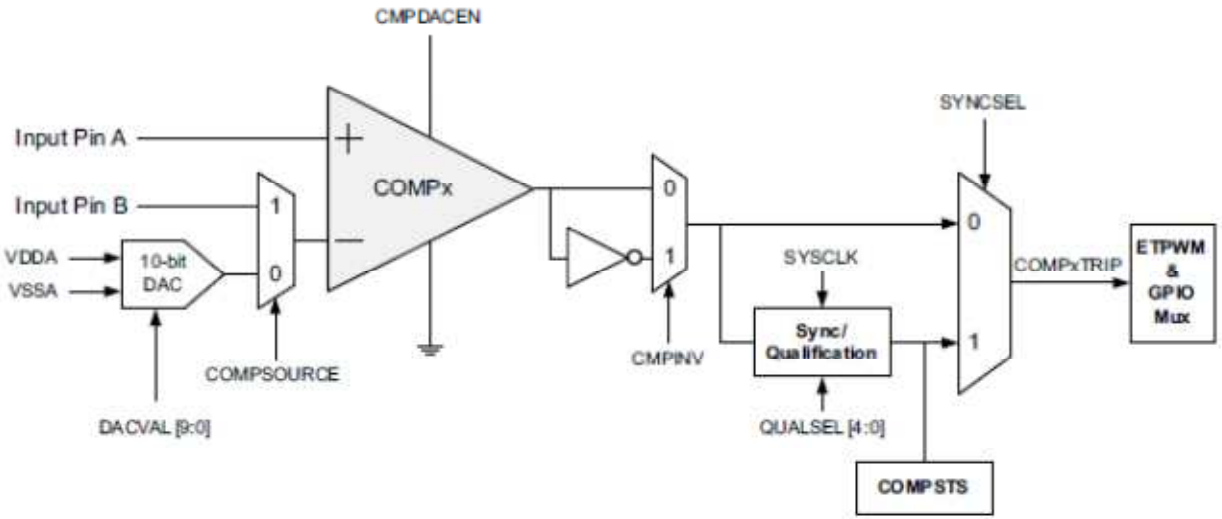
- Simultaneous Sampling Mode:
  - Decrease the delay between two measurements
    - Two consecutive SOC (0-1, 2-3, …) are measured at the same time

ADC Peripheral Module

- The End Of Conversion (EOC) pulse can occur at the beginning or the end of the conversion.
- The ADC contains 9 interrupts that can be triggered by the EOC signals
  - INTSELxNy register
    - INTxSEL or INTySEL
    - INTxCONT or INTyCONT
    - INTxE or INTyE

Comparator

- One or two internal “analog” comparators
- Use two external analog inputs or only one if the 10-bit DAC is used
- The output can be used synchronously or asynchronously
- The output can be applied to ePWM, Trip-Zone modules and to the GPIO multiplexer



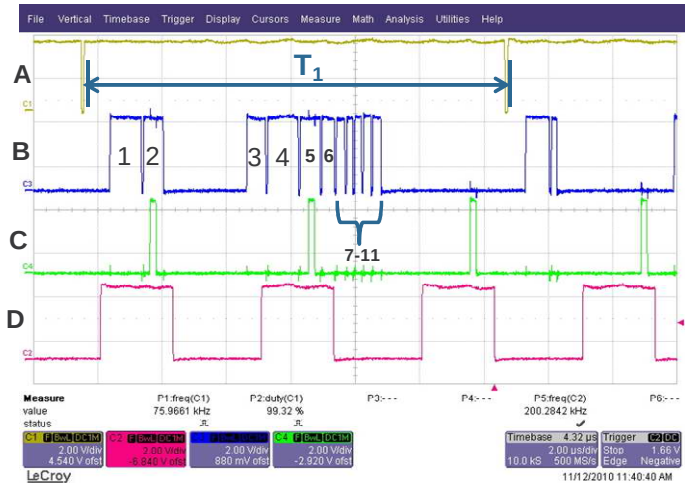
Exercise 1

- Using **Lab3** project, calculate the mean value of a PWM signal (filtering a PWM)
- Practice:
  - ADC
  - Interrupts
  - ePWM
- Tasks:
  - Modify ePWM1A frequency and adjust the number of points to be acquired. Example: create an array of 10 points for frequencies of 50kHz (ePWM2) and 5kHz (or 10kHz) (ePWM1)
  - <u>Write the code of the filter in the project file “DefaultIsr_3_4.c”:</u>
    - Measure X times in a period of ePWM1 and store in an array o x values
    - Obtain the “duty cycle” obtaining the mean of the measured values once the array is written with the new period values

Exercise 2

- Obtain the computation time of a task and the location in the period using an auxiliary signal:
  - Enable a GPIO signal
  - Set GPIO to high-state, then create an operation with integer data and finally set again to low-state.
  - Check with the oscilloscope the length of the task
  - Now define floating point data and do the same operation
  - Check the length in the oscilloscope the length of the task

- To enable a GPIO signal:
  - In GPIO.c, set the GPAMUX (or GPBMUX) properly
  - Configure the GPIOx as an output



Exercise 3

- Create a pulse of 500ms width on GPIO34 – Output can be seen also in the LED2 of the board (ON for low level of GPIO34)
- Practice:
  - GPIO
  - Interrupts
- Tasks:
  - In the file “DefaultIsr_3_4.c” add the code to control the GPIO34
    - Create a “GPIO34_count” counter
    - Calculate the number of interrupts to achieve
    - Toggle (change the value) of GPIO34 signal
      - GpioDataRegs.GPBTOGGLE.bit.GPIO34=1

Exercise 4

- Obtain the waveforms of two PWM modules:
  - Use of Action Qualifier (AQ) submodule of both ePWM modules
  - Information about the proper values of the registers in the ePWM module datasheet

Exercise 5

- Use of Dead-band time: Generate complementary signals with dead-band time (active high complementary)
- Generate other type of complementary signals
- Practice:
  - Dead-band time submodule configuration
- It is also possible to generate the complementary signal by using the registers of the PWM (without using Dead-band time control)

Exercise 6

- Synchronize two PWM modules (useful for obtain more accuracy in the duty cycle measurement in the exercise 3):
  - Synchronization of modules ePWM1 and ePWM2
  - Use of Time-Base module registers:
    - TBCTL.SYNCOSEL
    - TBCTL.PHSEN
  - Implement ePWM2 with a phase delay of 180° to ePWM1

Provided material

- Texas Instruments Piccolo one day tutorial (4 Labs)
  - Includes C28x1DayWorkshop file
- Piccolo main datasheets:
  - Piccolo Microcontrollers 2802X: TMS320F28027
  - Enhanced Pulse Width Modulator: Piccolo ePWM
  - Analog to Digital Converter: Piccolo ADC
  - Code Composer Studio Guide
  - Piccolo System Control and Interrupts
- Documentation folder

CEI UPM | Centro de Electrónica Industrial

Digital Control of Power Converters: Piccolo Microcontroller Training Class III

ceiupm.es

Universidad Politécnica de Madrid

POLITÉCNICA

Planning of the course

1 Introduction + assignment	February 1rd	Óscar	Seminar room
2 DSP Piccolo training class I	February 8th	Dani	Computers
3 Design of regulators I	February 15th	Jesús	Computers
4 DSP Piccolo training class II	February 22th	Dani	Computers
5 Hardware issues: DPWM & ADC	March 1nd	Óscar	Seminar room
6 Design of regulators II	March 8th	Jesús	Computers
7 DSP Piccolo training class III	April 5th	Dani	Computers
8 Advanced Simulink design	**April 19th**	Jesús	Computers
9 Applications and ideas	**April 26th**	Óscar	Seminar room
10 Review of assignment	**May 10th**	All	
11 Test & Final review of assignment	**May 17th**		

Schedule of Training Class III

1. Practical exercises
2. Theory: GPIO and Interrupts
3. IQMath → How to configure
4. Two phase synchronous buck converter PCB test bench for the assignment
5. Program structure for the assignment

General Purpose Input-Output (GPIO)

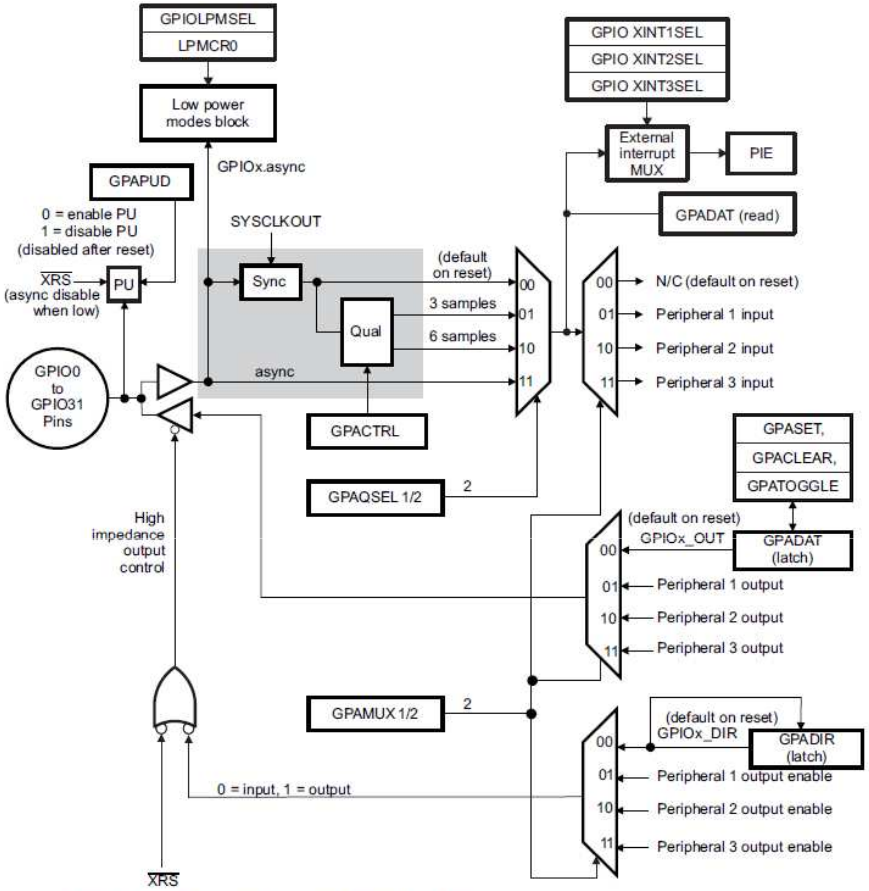
- GPIO
  - GPIOs located in two ports: A and B
  - *Port A: GPIO0-GPIO31*
    - GPAMUX1 (GPIO 0-GPIO 15)
    - GPAMUX2 (GPIO 16- GPIO 31)
  - *Port B: GPIO32-GPIO38+AIO0-AIO15*
    - GPBMUX1 (GPIO 32-GPIO38)
    - AIOMUX1 (AIO0-AIO15)
- Each GPIO has a two-bit register to:
  - Enable the GPIO
  - Enable any of the other peripheral functionalities (up to three): GpioCtrlRegs.GPAMUX1.bit.GPIO6=

    0: GPIO

    1: EPWM4A

    2: EPWMSYNCI

    3: EPWMSYNCO



A GPxDAT latch/read are accessed at the same memory location.

General Purpose Input-Output (GPIO)

- To use a GPIOx signal:
  - 1.- Assign the proper value in the correspondent GPIO PORT and MUX
  - 2.- Define it as an INPUT/OUTPUT → GPIOxDIR (x=A/B)
  - 4.- Used as an output → To change the GPIO signal value there are two options
    - Directly use GPxDAT
      - Example: GpioDataRegs.GPADAT.bit.GPIOx
    - Use the instructions GPxSET, GPxCLEAR and GPxTOGGLE
      - Example: GpioDataRegs.GPASET.bit.GPIO6
  - 5.- Used as an output → To change the GPIO signal value there are two options

Interrupts

- 96 sources of interrupts
- They are located in 12 groups of 8 interrupts each.
- When the interrupt INTx.y is generated, it is connected to PIE module
- Peripheral Interrupt Expansion (PIE) is used to deal efficiently with a big quantity of interrupts.
- Each group has an associated INTx signal

	INTx.8	INTx.7	INTx.6	INTx.5	INTx.4	INTx.3	INTx.2	INTx.1
INT1	WAKEINT	TINT0	ADCINT9	XINT2	XINT1		ADCINT2	ADCINT1
INT2					EPWM4_TZINT	EPWM3_TZINT	EPWM2_TZINT	EPWM1_TZINT
INT3					EPWM4_INT	EPWM3_INT	EPWM2_INT	EPWM1_INT
INT4								ECAP1_INT
INT5								
INT6							SPITX INTA	SPIRX INTA
INT7								
INT8							I2CINT2A	I2CINT1A
INT9							SCITX INTA	SCIRX INTA
INT10	ADCINT8	ADCINT7	ADCINT6	ADCINT5	ADCINT4	ADCINT3	ADCINT2	ADCINT1
INT11								
INT12								XINT3

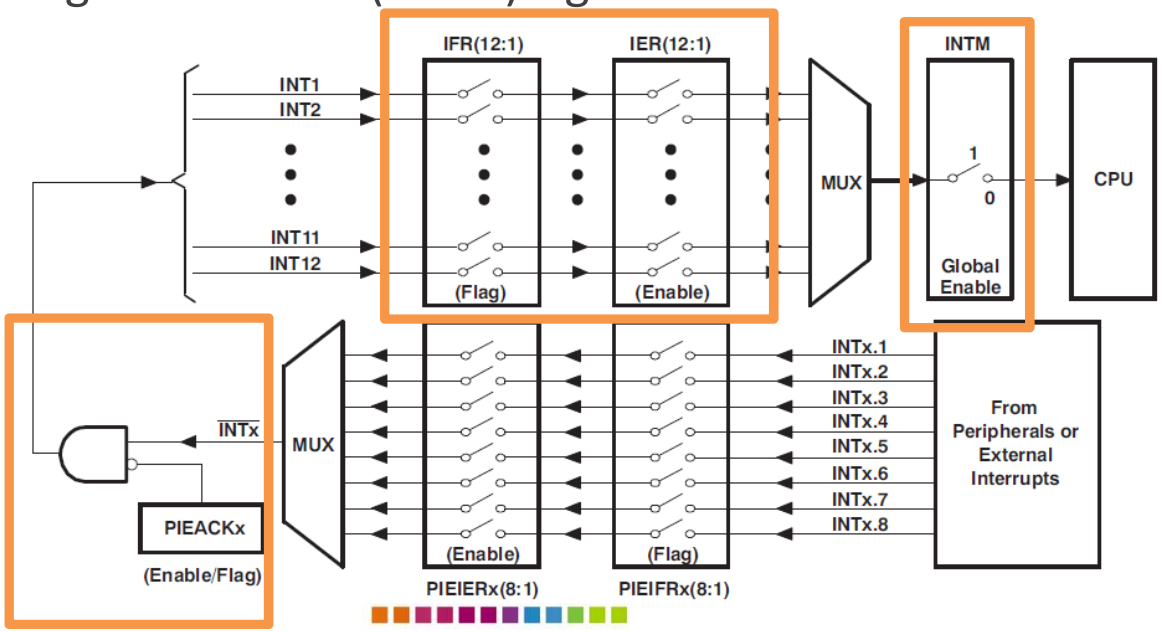
CPU

or

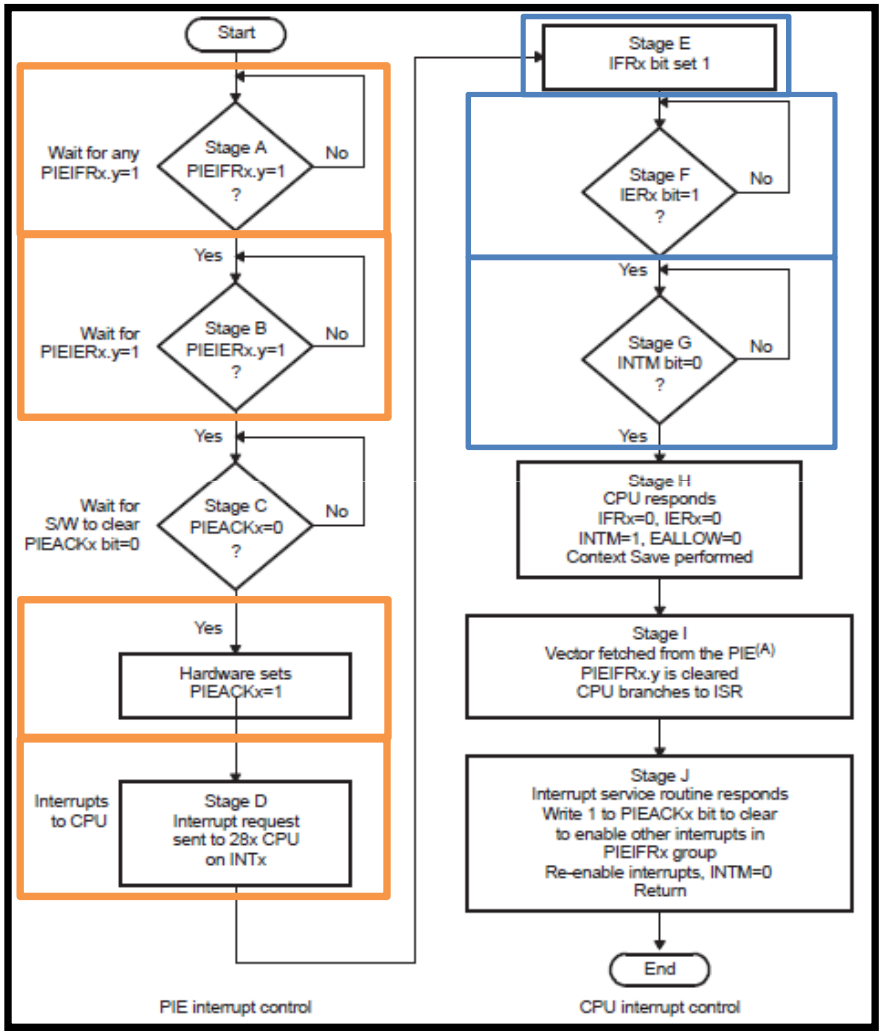
PIE Interrupt Assignment Table

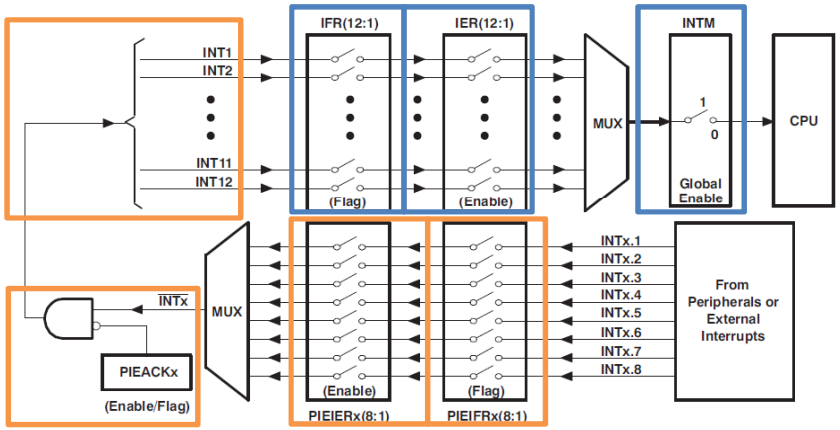
Interrupts

- The INTx signals are enabled by an PIEACKx signal: This flag has to be cleared <u>manually</u> inside the interrupt to enable the next interrupt event.
  - Example: PieCtrlRegs.PIEACK.bit.ACK1=1; writing “1” will enable INT1
- The interrupt flags within the peripheral registers must be manually cleared:
  - Example: AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
- Next step is IFR/IER (Interrupt Flag/Enable Register):
  - INTx generates IFR signal and, if correspondent IER is enabled, the interrupt reaches the output MUX (manages priority between interrupt)
- Finally, there is the global enable (INTM) signal



Interrupts


Start
Wait for any PIEIFRx.y=1
Stage A PIEIFRx.y=1 ?
No
Yes
Wait for PIEIERx.y=1
Stage B PIEIERx.y=1 ?
No
Yes
Wait for S/W to clear PIEACKx bit=0
Stage C PIEACKx=0 ?
No
Yes
Hardware sets PIEACKx=1
Interrupts to CPU
Stage D Interrupt request sent to 28x CPU on INTx
PIE interrupt control
Stage E IFRx bit set 1
Stage F IERx bit=1 ?
No
Yes
Stage G INTM bit=0 ?
No
Yes
Stage H CPU responds IFRx=0, IERx=0 INTM=1, EALLOW=0 Context Save performed
Stage I Vector fetched from the PIE(A) PIEIFRx.y is cleared CPU branches to ISR
Stage J Interrupt service routine responds Write 1 to PIEACKx bit to clear to enable other interrupts in PIEIFRx group Re-enable interrupts, INTM=0 Return
End
CPU interrupt control


INT1
INT2
INT11
INT12
IFR(12:1)
(Flag)
IER(12:1)
(Enable)
MUX
INTM
1
0
Global Enable
CPU
INTx
PIEACKx
(Enable/Flag)
MUX
(Enable)
PIEIERx(8:1)
(Flag)
PIEIFRx(8:1)
INTx.1
INTx.2
INTx.3
INTx.4
INTx.5
INTx.6
INTx.7
INTx.8
From Peripherals or External Interrupts

IQMath

- IQMath Library is collection of highly optimized and high precision mathematical function library for the calculation of floating point algorithms into fixed point code.
- Used when optimal execution speed & high accuracy is critical.
- It will be used in the assignment to reduce the computation time of the control loops
- It is modified the definition of the coefficients and the operations to use IQMath to decrease the computation time

Data Type	Range		Resolution/Precision
	Min	Max	
_iq30	-2	1.999 999 999	0.000 000 001
_iq29	-4	3.999 999 998	0.000 000 002
_iq28	-8	7.999 999 996	0.000 000 004
_iq27	-16	15.999 999 993	0.000 000 007
_iq26	-32	31.999 999 985	0.000 000 015
_iq25	-64	63.999 999 970	0.000 000 030
_iq24	-128	127.999 999 940	0.000 000 060
_iq23	-256	255.999 999 981	0.000 000 119
_iq22	-512	511.999 999 762	0.000 000 238
_iq21	-1024	1023.999 999 523	0.000 000 477
_iq20	-2048	2047.999 999 046	0.000 000 954
_iq19	-4096	4095.999 998 093	0.000 001 907
_iq18	-8192	8191.999 996 185	0.000 003 815
_iq17	-16384	16383.999 992 371	0.000 007 629
_iq16	-32768	32767.999 984 741	0.000 015 259
_iq15	-65536	65535.999 969 482	0.000 030 518
_iq14	-131072	131071.999 938 965	0.000 061 035
_iq13	-262144	262143.999 877 930	0.000 122 070
_iq12	-524288	524287.999 755 859	0.000 244 141
_iq11	-1048576	1048575.999 511 719	0.000 488 281
_iq10	-2097152	2097151.999 023 437	0.000 976 563
_iq9	-4194304	4194303.998 046 875	0.001 953 125
_iq8	-8388608	8388607.996 093 750	0.003 906 250
_iq7	-16777216	16777215.992 187 500	0.007 812 500
_iq6	-33554432	33554431.984 375 000	0.015 625 000
_iq5	-67108864	67108863.968 750 000	0.031 250 000
_iq4	-134217728	134217727.937 500 000	0.062 500 000
_iq3	-268435456	268435455.875 000 000	0.125 000 000
_iq2	-536870912	536870911.750 000 000	0.250 000 000
_iq1	-1073741824	1 073741823.500 000 000	0.500 000 000

Main IQMath Operations

- Variable definition in iqX:
  - _iqX var1;
- Variable assignment in iqX:
  - var1 = _IQX(0.00010717497823597);
- Multiplications of an IQX data with a 32bit integer:
  - var2=_IQXmpyI32(var3 , var4);
- Switch from IQX format to IQY format:
  - _IQXtoIQY(var5);
- Change of data format from IQX data to integer:
  - var6= _IQXint(var7);

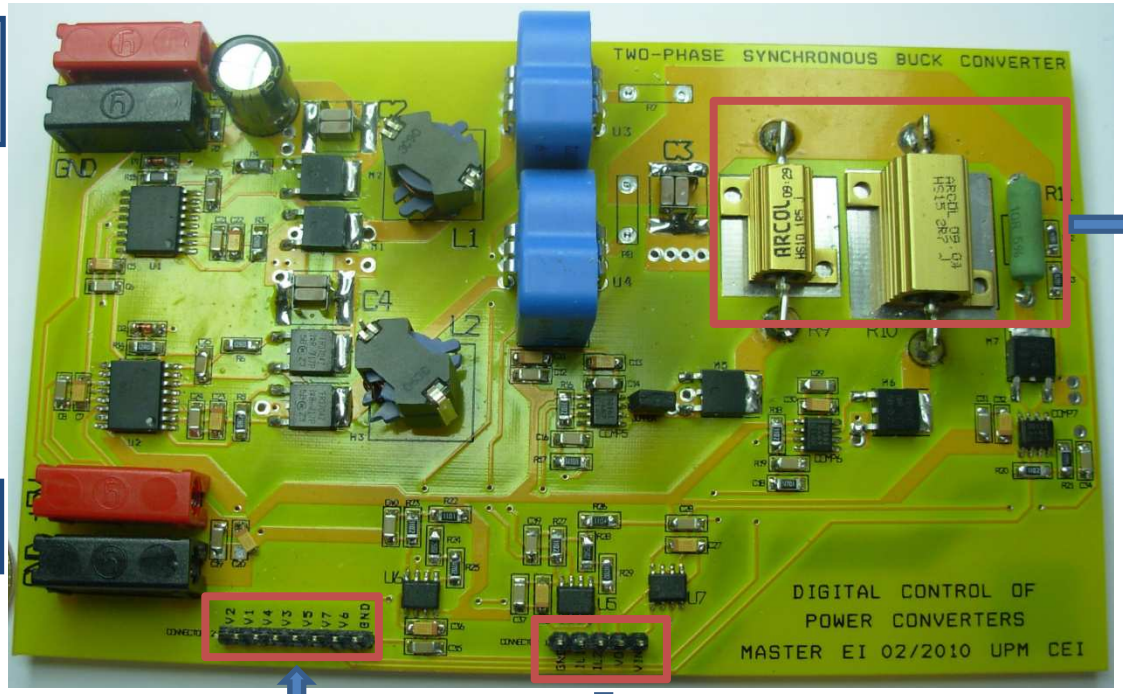
IQMath configuration

- To configure IQMath:
  - Copy the library “IQMath.lib” and the header file “**IQmathLib.h**” in your computer and ensure they are in an included folder
  - Substitute the file “**Lab_2_3.cmd**” with the code provided in “**Lab_2_3.txt**” (memory assignment for IQMath)
  - All the necessary files are in “..\Digital Control of Power Converters 2013/Code for the assignment” of servidordiscos\alumnos\MASTER EI folder

Two-phase synchronous buck converter

Converter supply voltage input (+12V)

Auxiliary supply voltage (+5V)



Loads (1R5, 2R7, 10R)

Input control signals

Measurements to ADC

- Two-phase synchronous buck converter
- Nominal operation (12V to 3.3V) (Around 12W at full load)
- Each load has a switch in series that is driven from de Piccolo MCU

Two-phase synchronous buck converter

- The schematic circuit of the converter can be found in the "SERVIDORDISCOS" common server, in the folder MASTER EI\Digital Control \ Digital Control of Power Converters 2013\Schematic Digital Control Buck PCB
- Power converter is provided soldered but main components (inductors, output capacitor) have to be measured to obtain a model as accurate as possible
- Current sensor provides a voltage offset → Measurement is processed to adjust the gain to take advantage of the ADC measurement range
- Converter can operate up to 5V at the output but <u>for voltages higher than 3.3V the 1R5 load has to be disconnected</u> (a jumper available to ensure it is not connected in any condition)
- There are three comparators to drive the MOSFETs that switch ON/OFF the loads
- The input voltage of the auxiliary power supply should not higher than 5.0V and 12V in the case of the main voltage supply

Modifications to be done in the program

- Main structure can be taken from Lab3 used for the training lessons, but some modifications have to be done
- The files with the necessary code can be found in the folder "..\Digital Control of Power Converters 2012/Base code for the assignment Digital Control 2012"
- The interrupt file "**DefaultIsr_3_4.c**" can be updated from the one used in the assignment. A guide of the program can be found in the file "**DefaultIsr_3_4.txt**"
- Modifications in the ADC converter configuration:
  - As we will measure more signals than in the tutorial, some registers have to be modified accordingly (especially the EOC that triggers the used interrupt ADCINT1)