Digital Control of Power Converters: Piccolo Microcontroller Training Class I
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

Main Characteristics:
- 60MHz (10 MHz internal clock), 32-bit fixed point performance core
- 12-bit ADC (16 Channels); ADC Resolution: 0V-3.3V
- 8 PWMs (4 HRPWM)
- IC USB-UART/FIFO Interface
- 2 Comparators
- 3 Timers
Digital controller: Piccolo 28069

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

---

Main Characteristics:
- 90MHz
- 32-bit fixed point performance core
- 12-bit ADC (16 Channels); ADC Resolution: 0V-3.3V
- 16 PWMs (8 HRPWM)
  - Up to 150-ps high resolution ePWM
- 3 Comparators
- 3 Timers

---

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADC-A6</td>
</tr>
<tr>
<td></td>
<td>COMP3(+VE)</td>
</tr>
<tr>
<td>2</td>
<td>ADC-A2</td>
</tr>
<tr>
<td></td>
<td>COMP1(+VE)</td>
</tr>
<tr>
<td>3</td>
<td>ADC-A0</td>
</tr>
<tr>
<td>4</td>
<td>3V3</td>
</tr>
<tr>
<td>5</td>
<td>ADC-A4</td>
</tr>
<tr>
<td></td>
<td>COMP2(+VE)</td>
</tr>
<tr>
<td>6</td>
<td>ADC-B1</td>
</tr>
<tr>
<td>7</td>
<td>EPWM-4B</td>
</tr>
<tr>
<td></td>
<td>GPIO-07</td>
</tr>
<tr>
<td>8</td>
<td>TZ1</td>
</tr>
<tr>
<td></td>
<td>GPIO-12</td>
</tr>
<tr>
<td>9</td>
<td>SClA</td>
</tr>
<tr>
<td></td>
<td>GPIO-33</td>
</tr>
<tr>
<td>10</td>
<td>ADC-B0</td>
</tr>
<tr>
<td></td>
<td>COMP(-VE)</td>
</tr>
<tr>
<td>11</td>
<td>EPWM-4A</td>
</tr>
<tr>
<td></td>
<td>GPIO-06</td>
</tr>
<tr>
<td>12</td>
<td>ADC-A1</td>
</tr>
<tr>
<td>13</td>
<td>SDAA</td>
</tr>
<tr>
<td></td>
<td>GPIO-32</td>
</tr>
<tr>
<td>14</td>
<td>ADC-B0</td>
</tr>
<tr>
<td>15</td>
<td>EPWM-3B</td>
</tr>
<tr>
<td></td>
<td>GPIO-05</td>
</tr>
<tr>
<td>16</td>
<td>5V0</td>
</tr>
<tr>
<td></td>
<td>(Unused by Default)</td>
</tr>
<tr>
<td>17</td>
<td>EPWM-1A</td>
</tr>
<tr>
<td></td>
<td>GPIO-00</td>
</tr>
<tr>
<td>18</td>
<td>ADC-B4</td>
</tr>
<tr>
<td></td>
<td>COMP2(-VE)</td>
</tr>
<tr>
<td>19</td>
<td>EPWM-3A</td>
</tr>
<tr>
<td></td>
<td>GPIO-04</td>
</tr>
<tr>
<td>20</td>
<td>SPISOMIA</td>
</tr>
<tr>
<td></td>
<td>GPIO-17</td>
</tr>
<tr>
<td>21</td>
<td>EPWM-1B</td>
</tr>
<tr>
<td></td>
<td>GPIO-01</td>
</tr>
<tr>
<td>22</td>
<td>ADC-A5</td>
</tr>
<tr>
<td>23</td>
<td>EPWM-2B</td>
</tr>
<tr>
<td></td>
<td>GPIO-03</td>
</tr>
<tr>
<td>24</td>
<td>SPISOMOA</td>
</tr>
<tr>
<td></td>
<td>GPIO-16</td>
</tr>
<tr>
<td>25</td>
<td>SPISTEA</td>
</tr>
<tr>
<td></td>
<td>GPIO-19</td>
</tr>
<tr>
<td>26</td>
<td>ADC-B2</td>
</tr>
<tr>
<td></td>
<td>COMP1(-VE)</td>
</tr>
<tr>
<td>27</td>
<td>EPWM-2A</td>
</tr>
<tr>
<td></td>
<td>GPIO-02</td>
</tr>
<tr>
<td>28</td>
<td>GND</td>
</tr>
<tr>
<td>29</td>
<td>SPICLKA</td>
</tr>
<tr>
<td></td>
<td>GPIO-18</td>
</tr>
<tr>
<td>30</td>
<td>GPIO-31</td>
</tr>
<tr>
<td></td>
<td>(LED)</td>
</tr>
<tr>
<td>31</td>
<td>PWM1A-DAC</td>
</tr>
<tr>
<td></td>
<td>(Filtered)</td>
</tr>
<tr>
<td>32</td>
<td>GND</td>
</tr>
</tbody>
</table>
MCU Architecture

Program Bus

- Sectored Flash
- RAM
- Boot ROM

CLA Bus

- ePWM
- eCAP
- eQEP
- 12-bit ADC
- Watchdog

Register Bus

- CPU
- 32-bit Auxiliary Registers
- 32x32 bit Multiplier
- R-M-W Atomic ALU

Real-Time JTAG Emulation

CLA Interrupt Manager

32-bit Timers

Data Bus

- CAN 2.0B
- I2C
- SCI
- SPI
- LIN
- GPIO
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
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

- Modularity of the code and Real-time mode: 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 project invokes all necessary tools to build the project (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)
- **Write the code of the filter in the project file “DefaultIsr_3_4.c”:**
  - Measure X times in a period of ePWM1 and store in an array of 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
  - Configure the ePWM2 to obtain the control pulses
  - 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)
Digital Control of Power Converters: Piccolo Microcontroller Training Class II
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)

![Diagram of ePWM module with submodules and connections]

---

![CEI UPM and Politécnica logos]

---

4
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 Synchronization
  - 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
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 2: ePWM1 configured up-down, single edge symmetric independent modulation A and B

EPwm1Regs.AQCTLA.bit.ZRO
EPwm1Regs.AQCTLA.bit.CBD
EPwm1Regs.AQCTLB.bit.ZRO
EPwm1Regs.AQCTLB.bit.PBD
ePWM Dead-band (DB) submodule

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

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

FED = DBFED × T_{TBCLK}/2
RED = DBRED × 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
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

<table>
<thead>
<tr>
<th>15</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>6</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGSEL</td>
<td>Reserved</td>
<td>CHSEL</td>
<td></td>
<td>ACQPS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R/W-0</td>
<td>R-0</td>
<td>R/W-0</td>
<td></td>
<td>R/W-0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ADC Clock</th>
<th>ACQPS</th>
<th>Sample Window</th>
<th>Conversion Time (13 cycles)</th>
<th>Total Time to Process Analog Voltage (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>60MHz</td>
<td>6</td>
<td>116.67ns</td>
<td>216.67ns</td>
<td>333.34ns</td>
</tr>
<tr>
<td>60MHz</td>
<td>8</td>
<td>150.00ns</td>
<td>216.67ns</td>
<td>366.67ns</td>
</tr>
<tr>
<td>60MHz</td>
<td>10</td>
<td>183.33ns</td>
<td>216.67ns</td>
<td>400.00ns</td>
</tr>
<tr>
<td>60MHz</td>
<td>14</td>
<td>250.00ns</td>
<td>216.67ns</td>
<td>466.67ns</td>
</tr>
<tr>
<td>60MHz</td>
<td>25</td>
<td>433.33ns</td>
<td>216.67ns</td>
<td>650.00ns</td>
</tr>
<tr>
<td>40MHz</td>
<td>6</td>
<td>175.00ns</td>
<td>325ns</td>
<td>500.00ns</td>
</tr>
<tr>
<td>40MHz</td>
<td>25</td>
<td>625ns</td>
<td>325ns</td>
<td>950.00ns</td>
</tr>
</tbody>
</table>

(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)
- **Write the code of the filter in the project file “DefaultIsr_3_4.c”:**
  - Measure X times in a period of ePWM1 and store in an array of x values
  - Obtain the “duty cycle” obtaining the mean of the measured values once the array is written with the new period values
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
Digital Control of Power Converters: Piccolo Microcontroller Training Class III
## Planning of the course

<table>
<thead>
<tr>
<th>Topic</th>
<th>Date</th>
<th>Instructor</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction + assignment</td>
<td>February 1rd</td>
<td>Óscar</td>
<td>Seminar room</td>
</tr>
<tr>
<td>2. DSP Piccolo training class I</td>
<td>February 8th</td>
<td>Dani</td>
<td>Computers</td>
</tr>
<tr>
<td>3. Design of regulators I</td>
<td>February 15th</td>
<td>Jesús</td>
<td>Computers</td>
</tr>
<tr>
<td>4. DSP Piccolo training class II</td>
<td>February 22th</td>
<td>Dani</td>
<td>Computers</td>
</tr>
<tr>
<td>5. Hardware issues: DPWM &amp; ADC</td>
<td>March 1nd</td>
<td>Óscar</td>
<td>Seminar room</td>
</tr>
<tr>
<td>6. Design of regulators II</td>
<td>March 8th</td>
<td>Jesús</td>
<td>Computers</td>
</tr>
<tr>
<td>7. DSP Piccolo training class III</td>
<td>April 5th</td>
<td>Dani</td>
<td>Computers</td>
</tr>
<tr>
<td>8. Advanced Simulink design</td>
<td><strong>April 19th</strong></td>
<td>Jesús</td>
<td>Computers</td>
</tr>
<tr>
<td>9. Applications and ideas</td>
<td><strong>April 26th</strong></td>
<td>Óscar</td>
<td>Seminar room</td>
</tr>
<tr>
<td>10. Review of assignment</td>
<td>May 10th</td>
<td>All</td>
<td></td>
</tr>
<tr>
<td>11. Test &amp; Final review of assignment</td>
<td>May 17th</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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: EPWMSYNCO
      ```
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 \(\rightarrow\) GPIOxDIR \((x=\text{A/B})\)
  4. Used as an output \(\rightarrow\) **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 \(\rightarrow\) **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
Interrupts

- The INTx signals are enabled by an PIEACKx signal: This flag has to be cleared manually 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

Stage A
IFR = 1

Stage B
PIE

Stage C
PEACK = 0

Stage D
Interrupt request sent to CPU on INTx

Stage E
IFRx bit set 1

Stage F
IER = 1

Stage G
INTM bit = 0

Stage H
CPU responds
IFR = 0, IER = 0
Context Save performed

Stage I
PIE(F) set 1
CPU branches to ISR

Stage J
Interrupt service routine responds
Write 1 to PEACKx bit to clear
Interrupt requests in PIE group
Re-enable interrupts, INTM = 0
Return

End

CPU interrupt control

PIE interrupt control
IQMath

- IQMath Library is a 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.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Range</th>
<th>Resolution/Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>-2</td>
<td>0.00000001</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0.00000002</td>
</tr>
<tr>
<td>10</td>
<td>0.0000</td>
<td>0.00000004</td>
</tr>
<tr>
<td>0</td>
<td>0.0000</td>
<td>0.00000007</td>
</tr>
<tr>
<td>-10</td>
<td>-0.5</td>
<td>0.00000015</td>
</tr>
<tr>
<td>-20</td>
<td>-1.0</td>
<td>0.00000040</td>
</tr>
<tr>
<td>-30</td>
<td>-1.5</td>
<td>0.00000081</td>
</tr>
<tr>
<td>-40</td>
<td>-2.0</td>
<td>0.00000162</td>
</tr>
<tr>
<td>-50</td>
<td>-2.5</td>
<td>0.00000325</td>
</tr>
<tr>
<td>-60</td>
<td>-3.0</td>
<td>0.00000650</td>
</tr>
<tr>
<td>-70</td>
<td>-3.5</td>
<td>0.00001300</td>
</tr>
<tr>
<td>-80</td>
<td>-4.0</td>
<td>0.00002550</td>
</tr>
<tr>
<td>-90</td>
<td>-4.5</td>
<td>0.00005000</td>
</tr>
<tr>
<td>-100</td>
<td>-5.0</td>
<td>0.00010000</td>
</tr>
<tr>
<td>-110</td>
<td>-5.5</td>
<td>0.00020000</td>
</tr>
<tr>
<td>-120</td>
<td>-6.0</td>
<td>0.00040000</td>
</tr>
<tr>
<td>-130</td>
<td>-6.5</td>
<td>0.00080000</td>
</tr>
<tr>
<td>-140</td>
<td>-7.0</td>
<td>0.00160000</td>
</tr>
<tr>
<td>-150</td>
<td>-7.5</td>
<td>0.00320000</td>
</tr>
<tr>
<td>-160</td>
<td>-8.0</td>
<td>0.00640000</td>
</tr>
<tr>
<td>-170</td>
<td>-8.5</td>
<td>0.01280000</td>
</tr>
<tr>
<td>-180</td>
<td>-9.0</td>
<td>0.02560000</td>
</tr>
<tr>
<td>-190</td>
<td>-9.5</td>
<td>0.05120000</td>
</tr>
<tr>
<td>-200</td>
<td>-10.0</td>
<td>0.10240000</td>
</tr>
<tr>
<td>-210</td>
<td>-10.5</td>
<td>0.20480000</td>
</tr>
<tr>
<td>-220</td>
<td>-11.0</td>
<td>0.40960000</td>
</tr>
<tr>
<td>-230</td>
<td>-11.5</td>
<td>0.81920000</td>
</tr>
<tr>
<td>-240</td>
<td>-12.0</td>
<td>1.63840000</td>
</tr>
<tr>
<td>-250</td>
<td>-12.5</td>
<td>3.27680000</td>
</tr>
<tr>
<td>-260</td>
<td>-13.0</td>
<td>6.55360000</td>
</tr>
<tr>
<td>-270</td>
<td>-13.5</td>
<td>13.10720000</td>
</tr>
<tr>
<td>-280</td>
<td>-14.0</td>
<td>26.21440000</td>
</tr>
<tr>
<td>-290</td>
<td>-14.5</td>
<td>52.42880000</td>
</tr>
<tr>
<td>-300</td>
<td>-15.0</td>
<td>104.85760000</td>
</tr>
<tr>
<td>-310</td>
<td>-15.5</td>
<td>209.71560000</td>
</tr>
<tr>
<td>-320</td>
<td>-16.0</td>
<td>419.43120000</td>
</tr>
<tr>
<td>-330</td>
<td>-16.5</td>
<td>838.86240000</td>
</tr>
<tr>
<td>-340</td>
<td>-17.0</td>
<td>1677.72480000</td>
</tr>
<tr>
<td>-350</td>
<td>-17.5</td>
<td>3355.44960000</td>
</tr>
<tr>
<td>-360</td>
<td>-18.0</td>
<td>6710.89920000</td>
</tr>
<tr>
<td>-370</td>
<td>-18.5</td>
<td>13421.79840000</td>
</tr>
<tr>
<td>-380</td>
<td>-19.0</td>
<td>26843.59680000</td>
</tr>
<tr>
<td>-390</td>
<td>-19.5</td>
<td>53687.19360000</td>
</tr>
<tr>
<td>-400</td>
<td>-20.0</td>
<td>107374.18230000</td>
</tr>
</tbody>
</table>
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 = _IQXmpyl32(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

- 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 for voltages higher than 3.3V the 1R5 load has to be disconnected (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).