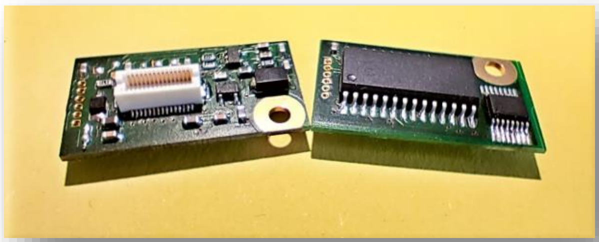
PIC32MK and PICMX: interface tests

Control Driver (PIC32MK)


PIC32MX250 test board

1 Revision Description

Revision R01-January 2018
- XC32 compiler 1.44 version
- External Quartz 12 Mhz
- MPLAB X IDE 4.3
- PicKit programmer
- UART1 pin configuration (C7 and C6)

Revision R02-Decembre 2021
- XC32 compiler 2.5 version
- External Quartz 12 Mhz
- MPLAB X IDE 5.5
- UART1 pin configuration (C7 and C6)
- Snap-Sn as programmer

Revision R03-Decembre 2021
- XC32 compiler 2.5 version
- FRC internal clock
- MPLAB X IDE 5.5
- UART1 pin configuration (G8 and G6)
- Snap-Sn as programmer

Revision R04-January 2022 (cleaning, simplification)
- XC32 compiler 2.5 version
- FRC internal clock
- MPLAB X IDE 6.0
- UART1 pin configuration (G8 and G6)
- Snap-Sn as programmer

Revision R05-April 2022 (cleaning, simplification)
XC32 compiler 2.5 version
FRC internal clock
MPLAB X IDE 6.0
UART1 pin configuration to be adapted for given hardware configuration
Snap-Sn as programmer
PIC32MX added

2 Few comments for all examples:

a. The main idea is to share very simple, short (one page) commented and complete examples.
b. All tests are based on PIC32MK1024MCF064 but they could be easy adapted for different devices
c. Check link to <p32mk1024mcf064.h>
d. Pay attention on the UART installation: pins initialization and channel number (printf on UART1).
e. The UART and its echo **(included into some examples)** is applying to communicate with test and verify if the PIC is working.
f. Some preliminary and verification tests are commented in /* */ (example UART)
g. If you have any comments, improvements, please lets help me
h. I'm using: "REGbits" solution, REGSET/REGCLR or own macro CLRBIT/SETBIT.
i. Some reference to Microchip's documentation are based on: PIC32MK GENERAL PURPOSE AND MOTOR CONTROL (GP/MC) FAMILY; ref: DS60001402D document

3 Test description

3.1 PIC32MK-DMA-R04_Receiver Test:

a. **Short description**
This test is "byte by byte" UART receiver based on the DMA access. It is applying to fill internal buffer. The example could be adapted for different applications.
b. After Booting hit different keys on the keyboard to fill the buffer. After CR, its content will be displayed

3.2 PIC32MK-DMA-R04_TransmitterTest:

a. **Short description**
One message is transmitted by the UART1 (terminal) when you hit 's" key. I'm applying it to interface the user sending stored strings. Observe that this message is sent only up to defined character '\n'.

3.3 PIC32MK-DMA-UART-R04:

a. Short description
One message stored via UART's receiver is displayed when you hit 'CR' key. This example is based on examples 3.1 and 3.2.

3.4 PIC32MK-SPI2-RO2_Test:

a. **Short description**
In this example any pushed key on the keyboard is sent by SPI2. You need to adapt pins and SPI's module number for your application. I'm using it to control 16 digital I/O implemented on the applied CPLD (internal shifter and latch) (see pdf). You can connected MISO and MOSI pins to implement a communication loop

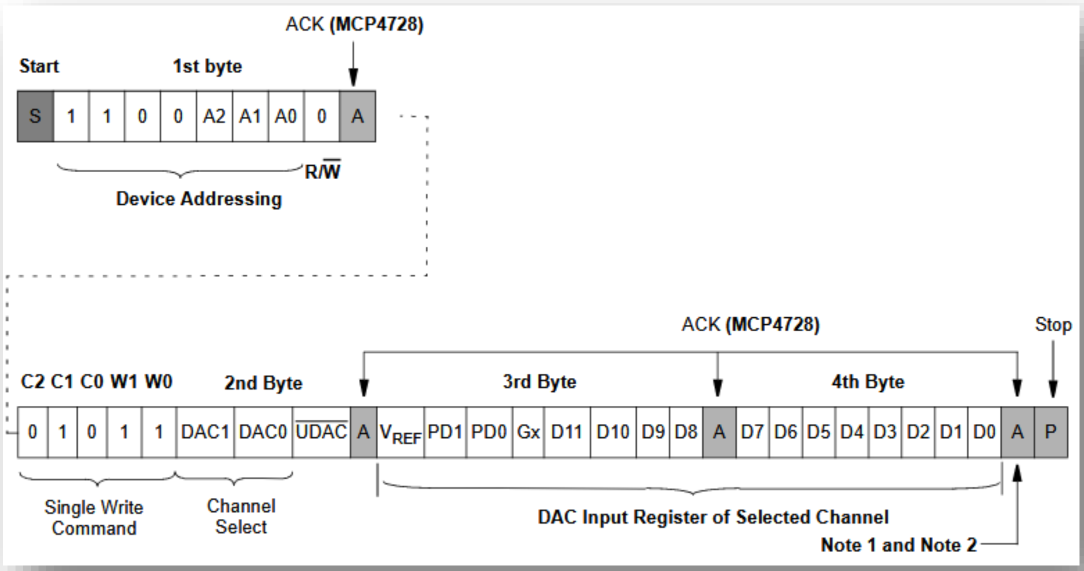
b. Pin C8 => MISO, D5 => MOSI, B6 => CLK

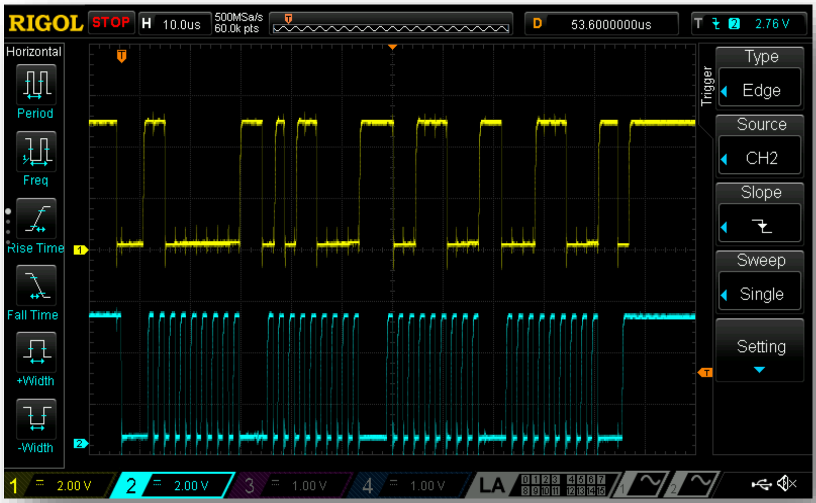
3.5 PIC32MK-I2C-RO2_Test:

a. **Short description**
That was certainly more complicated example. There are not any documentation (empty chapter) and any description. Finally, I've tested different channel and "discover" relation between one pair of pins and its number. I'm using this interface to configure my hardware after supply set-on. That for, I don't use **any interruption**. In any control loop only SPI should be applied (that is my opinion). The presented example interfaces excellent **Microchip's 12 bits DAC MCP4728** (nonvolatile), small footprint, low power, different configurations and buffered outputs). It is applied into my applications to define shifting for internal ADC. The example includes also BRG calculation for I2C which are "acceptable" by XC32.I've tried to be simpler **as possible**. This example can be extended to more complex procedures adding necessary **masks** and required sequences. It includes also while's loop **not blocking** implementation.

b. E12 and E13 as I2C pins

c. To test I2C interface you need hit a related command as: “1as” (1=> DAC channel 1, “a”=> data = 500, “s’ => send)



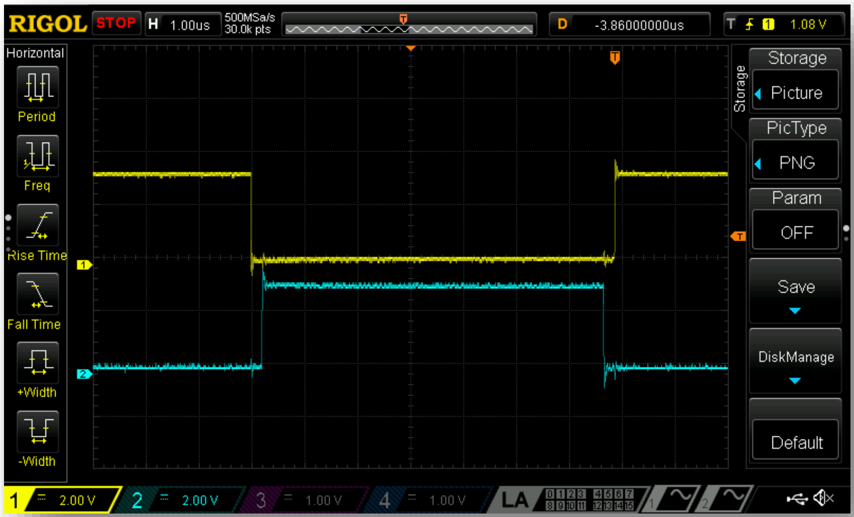


3.6 PIC32MK-ADC-RO3_Test :

a. I'd essentially HW problems essentially related to interface pre-amplifiers. But ....The new example is ready and additionally I didn't applied "REGbits" solutions. The main idea is to initialize all 0 to 5 and 7 ADC converters except ANEN and digital bits in the general initialization. The applied converters should be enabled only for given application. Anyway, example is fully commented and includes few printf to control some parameters during tests.
b. I'm using A0 to A5 pins.
c. The conversion start after hiting "C" key. Enjoy you !

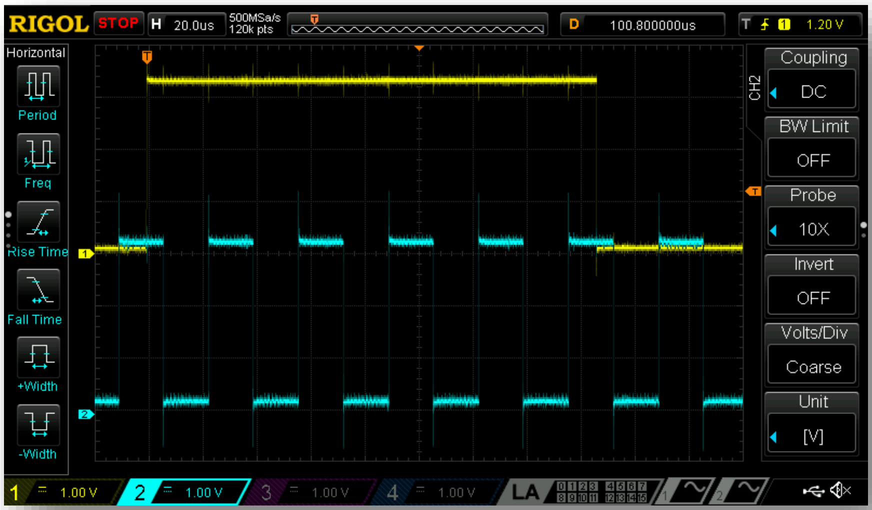
3.7 PIC32MK-PWM-RO3_Test :

a. That is very simple example for channel 1 as Complementary two pins PWM outputs. I've added Deat Time and its compensation. It is possible to change Duty Cycle hiting '1' to '9' key. Enjoy you !
b. Output pins are B14 and B15 which are not remapped.

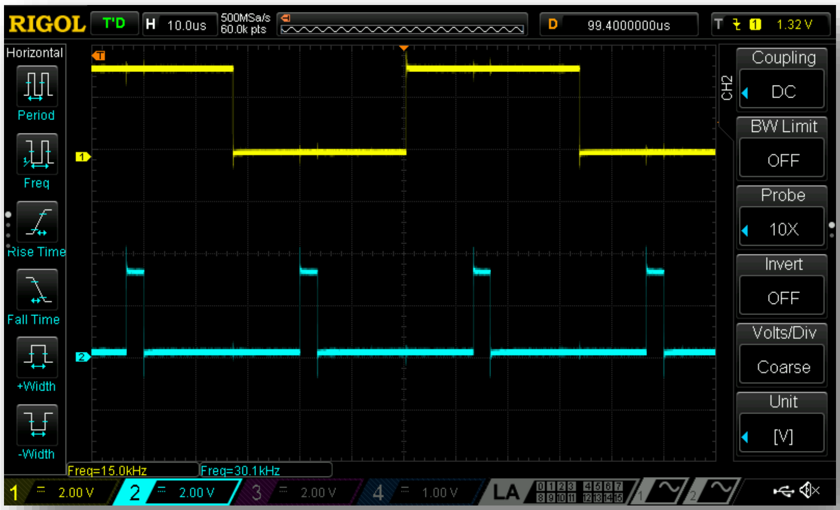


3.8 PIC32MK-PWM-INTERRUPTION-RO3_Test :

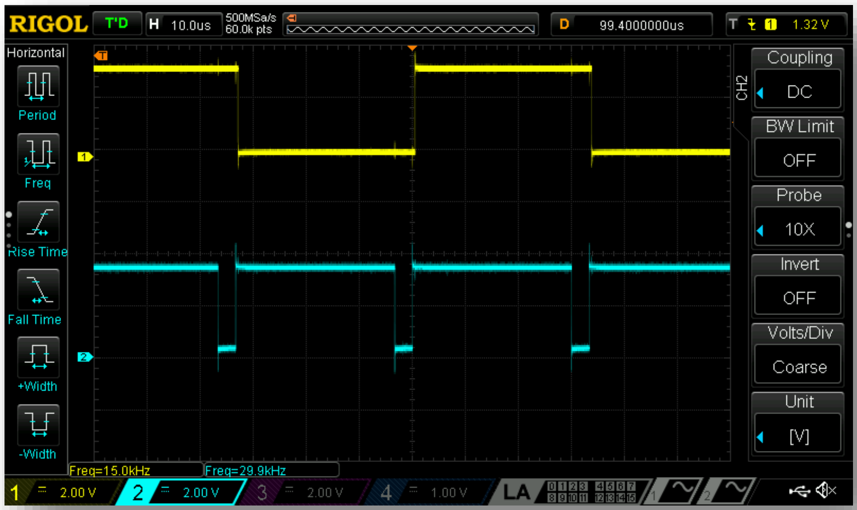
a. The main idea is to add ADC conversion in the middle of PWM step. To do it I've implemented the Trigger interruption service.
b. Additionally, to verify synchronization I've added two pins F0 and F1 to be connected with the oscilloscope.
c. Be careful: when you enable the PWM module SETBIT(PTCON,15); //! PWM module enable) the all unused PWM pins are overwritten (blocked). To have absolutely disable related PWM channels. For 64 pins packaging and F0 and F1:
 PMD4SET = 0xFFF00000; //! Disable channels PWM5 to PWM12 !!!!!!!!!
d. That is very simple example for channel 1 as Complementary two pins PWM outputs. I've added also Death Time and its compensation. Enjoy you !



TRGDIV1=4, Top=> FO toggled, Bottom=> PWM (50%)



TRGDIV1=0, Top=> FO toggled, Bottom=> PWM (90%)

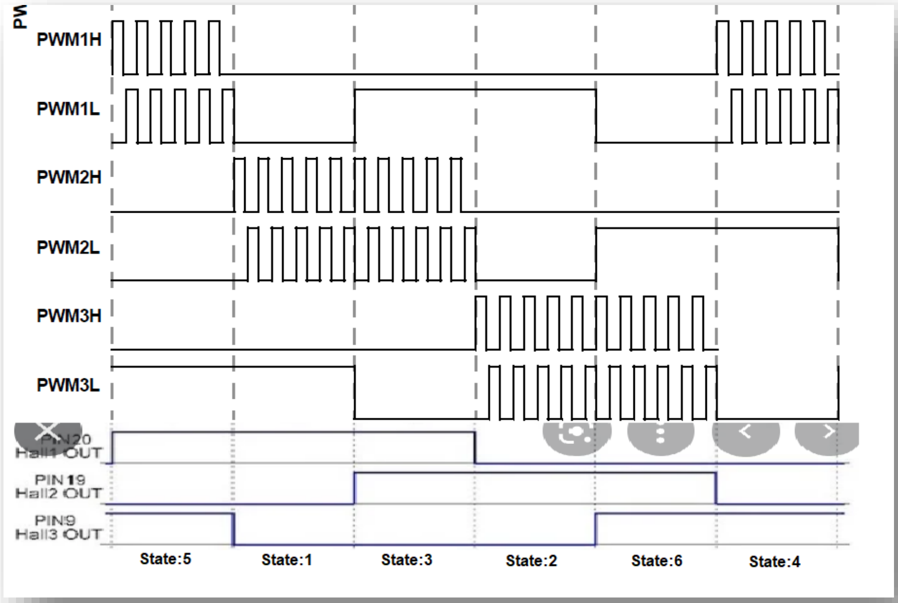


TRGDIV1=0, Top=> FO toggled, Bottom=> PWM (10%

3.9 PIC32MK-CN-IO-INTERUPT-PWM-BLDC-R03_Test :

a. **Change Notice (CN) pins:**
The CN pins provide PIC32 devices to generate interrupt requests to the processor in response to a change of state on selected input pins. I've applied that ability to generate the commutation for BLDC motor. In this case three Hall sensors (Open Collector) are connected to E12, E13 and E14 with pull-up resistors.

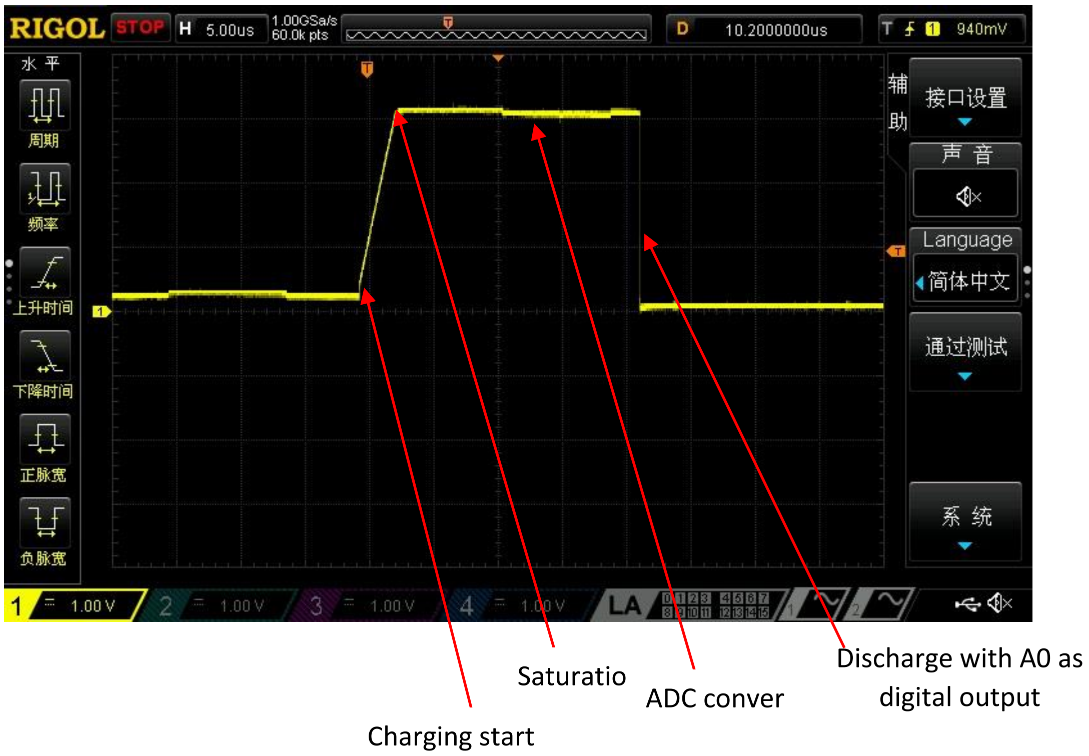
b. The implemented sequence of state are presented by following image:



c. Three PWMs (low and high) output are configured on B14/B15, B12/B13 and B10/B11 pins.
d. It is possible to modify Duty Cycle, Period, Death Times etc.
e. The ADC interface is presented only for PWM1 (See test Nr. 7)

3.10 PIC32MX-CTMU-ADC-R02:

a. **CTMU (Charge Time Measurement Unit:**
The CTMU is interesting unit and could be not essentially to interface Touch buttons but also to measure resistance or capacitance. Some sensors apply capacitor as intermediary value.
b. The available docs seems to be simple and evident but CTMU implementation could be complex for different devices as example PIC32MK familly.
c. The main difficulties is related to synchronize sequence between CTMU, current source and ADC control.
d. The small 10 pF capacitor is connected between A0 (pin 1) and Gnd.
e. You need to activate “c” key to start charging.
f. This example could be easily adapted to:
  1. Resistance measurement
  2. Also capacitance using timer to limit charging time and final voltage level
  3. I’ve tried to replace timer by comparator and internal Vref unit but the internal input impedance is so small that charging stop to be linear. In this case, the A0 was connected in parallel with comparator input.



RIGOL STOP H 5.00us 1.00GSa/s 60.0k pts D 10.2000000us T 1 940mV
1 1.00 V 2 1.00 V 3 1.00 V 4 1.00 V