



STATE MACHINE PART4

A Clock Calendar full project with the DS1307

PIC®, MPLAB®, PICKit3® and ICD3® are registered trademarks of Microchip Technology Inc®.
Proton Development Suite® or PDS® are a registered trademark of Crownhill Associates Limited®.

The project has been developed and written by Alberto Freixanet.
The document has been edited by John Drew.

Introduction:

We are going to study a full project demo made with my State Machine.
You have to forget to program in linear mode and think about the structured method.
I remember the principles of my State Machine:

- 1- This State Machine is a large loop that can never stop.
- 2- The State Machine must run at maximum speed to ensure the timing is correct. Take it into account when typing the code.
- 3- You cannot use the 'GoTo' command to a destination that is outside the static code area of a module.
- 4- Do not delay the main code by inserting other long codes, better to use the State Machine2 (SM2) in the background or split the code into several modules.
- 5- Use the SM2 to maximum when the SM1 is stopped by a virtual delay.
- 6- The SM allows separation of the whole program into small autonomous modules. If a single-module code fails, it could be a single problem with this same code or badly initialised parameters.

Characteristics of the project:

Clock Calendar clock with DS1307 I²C.

Contrast setting by software. (ST7036 LCD Shield)

Setting the clock: Date, Hour, minute.

Setting Alarm 1 daily: Hour, minute.

Setting Alarm 2 yearly: Date, Hour, minute.

Checking the pushbuttons.

I will describe the code in as much detail as I can so that the beginner can understand how this project is very different to linear programming.

INTERRUPT HANDLER:

The Timer2 and PR2 are used to generate an interrupt every 1mS, because it is easy to configure and does not need any code to reload the timer in the interrupt handler. Several timings have been prepared to generate future functions.

This program generates timings of 10mS and 1 Sec.

```
$if _defined(_Timer10mS) Or _defined(_Timer1S)  
...  
$endif
```

Every 10mS the PWM of the backlight is started.

```
' Start the PWM BackLight every 10mS  
High bLCD_BackLight  
Set F_PWMBackLight  
PWMBLCounter = PWMBLCounterValue
```

Every 500mS the special flag F_Timer500mS is toggled to generate a blinking flag.

```
Toggle F_Timer500mS ' 1 Hz Timer, to toggle some leds.
```

Several delays are prepared to perform functions for the SM or for the user.

```
$ifdef _SMVDelay  
' Delay 1mS for the States.  
If F_SMVDelay = 1 Then ' The VDelay is started.  
    Dec SMDelayCounter ' Decrements SMDelayCounter every 1mS.  
    If SMDelayCounter = 0 Then ' Check if delay counter = 0.  
        F_SMVDelay = 0 ' Delay END, clear Vdelay FLAG.  
    EndIf  
EndIf  
$endif
```

Every 1mS the PWM delay runs.

```
' Count down the PWM BackLight.  
If F_PWMBackLight = 1 Then  
    Dec PWMBLCounter ' Decrements the DelayCounter every 1mS.  
    If PWMBLCounter = 0 Then ' Check if delay counter = 0.  
        F_PWMBackLight = 0 ' Delay End, clear the delay FLAG.  
        Low bLCD_BackLight  
    EndIf  
EndIf
```

Every 1mS the TimeOut Delay could run.

```
' Count every 1mS.  
If F_TimeOutDelay = 1 Then ' The Delay is started.  
    Dec TimeOutCounter ' Decrements the DelayCounter every 1mS.  
    If TimeOutCounter = 0 Then ' Check if delay counter reaches 0.  
        F_TimeOutDelay = 0 ' Delay End, clear the delay FLAG.  
    EndIf  
EndIf
```

Interruption generated by the 1Hz signal of the DS1307.

```
' PORTA.4 TMR0 Interrupt
' The Timer0 interrupt is used at High to Low edge interrupt.
If F_TMR0IF = 1 Then
    F_TMR0IF = 0
    F_TMR0ON = 0
    Nop
    TMR0L = 255
    Nop
    F_TMR0ON = 1
    F_DS1307_1S = 1
EndIf
```

On the LCD board the 1 Hz signal is not connected to the PIC. There was no PORTB pin available to make an interrupt, but it could be connected to PORT4 corresponding to timer0 external input. By correctly setting the Timer0 with a preset value of 255, with falling edge of the input pin, a direct interruption can be obtained at each pulse of the 1 Hz signal.

STATE00: (Initialisation of parameters and/or Title or the project)

User command: **SetBackLightTime(Max)**

In this module some sentences are sent to the LCD that needs to be illuminated. This command activates the backlight without delay.

The AMI18 LCD Shield is a commercial product at www.picshop.nl, it is fully compatible with the Amicus18 Board. However the pin used for this function is PORTB.1. It is a badly chosen output because a PWM function cannot be used to vary the brightness of the LCD. The function P1C could be used but the Timer2 is busy performing the timing of the interruption of the SM. There is no other solution than to perform my own PWM function.

Two levels of brightness are really needed; One maximum and another one to set the minimum brightness for the LCD.

Using the available means, it is possible to realise a PWM with a resolution of 10 steps (between 3 and 4 bits of resolution) which is sufficient for our application.

Two codes are needed in the interrupt routine.

- Activate the backlight every 10mS (available in the SM)
- Set a value of a 1ms resolution delay to perform the PWM.

```
' Start the PWM BackLight every 10mS
High LCD_BackLight
Set F_PWMBackLight
PWMBLCounter = PWMBLCounterValue

' Count down the PWM BackLight.
If F_PWMBackLight = 1 Then
    Dec PWMBLCounter      ' Decrements the DelayCounter every 1mS.
    If PWMBLCounter = 0 Then ' Check if delay counter reaches 0.
        F_PWMBackLight = 0 ' Delay End, clear the delay FLAG.
        Low LCD_BackLight
```

```
EndIf
EndIf
```

Normally the order of the routines should be reversed in the interrupt routine, compensated by writing (BackLight Time = pTime + 1)

SetBackLightTime(Max) Macro:

```
$define SetBackLightTime(pTime) '
    $if pTime = Max                '
    BackLightTime = pTime + 1      '
    PWMBLCounterValue = 11        '
    $else                          '
    PWMBLCounterValue = 11        '
    $endif
```

If (pTime) value is different from the word "Max" the code corresponding to the time is not pasted in the code. This time value is no longer required to activate the backlight continuously.

The command activates the BackLight counter to 11. Since the counter has 10 stages to decrement, the PWMBLCounter could never reach 0 so the BackLight is always on.

SetBackLightTime(3)

The time counter is set to 3 seconds.

Library command: **HRSLStrg**(TXT0,AllChars,1)

This command is from the library of the State Machine available to the user of PDS.

All project texts are written at the end of the .bas file with a Cdata table but the HRSSOut command writes its table at the beginning of the program.

For this reason a specific macro has been written. Special functions have been added to this occasion. Calling a **LABEL** can save many bytes of code if it is used more than once.

TXT0: Label of the String to print.

AllChars: Print all the characters of the String (1 to 255).

1: Carriage Return + Line Feed number is sent to the terminal (0 to 255)

Example:

HRSLStrg(TXT0,AllChars,1)

It may be written more simply as **HRSLStrg**(TXT0, 1) if it is assumed that all characters are written.

But you cannot write more code on the same line as it would generate a compile error.

SM command: **NextStateDelay**(3000,IncState)

This command has been described in an earlier chapter. As there is no more code to execute, the information is given to the SM to go to the next state with a delay of 3000 mS.

User command: **LCD_Clear()**

This macro corresponds to the following code. It is equivalent to Cls.

Print \$FE,1

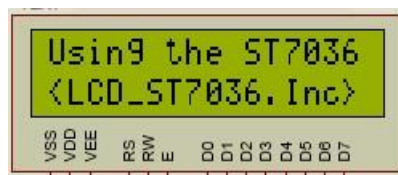
Library command: **PrintStrg(TXT0,1,1,AllChrs)**

This command performs the same function as above but applies to the Print command of the PDS.

Some parameters are initialised.

```
SM_UserFunctionFlags = 0
SM_UserSystemFlags = 0
AllButtonsBitsDefects = %11110000    ' NO button fails. Reset value.
ButtonFail = 0
```

STATE01: (Information of the project: LCD, Baud Rate,...)



Library command: **HRSE2PStrg(Address,AllChars,1)**

This command sends to the terminal the strings written by the **Edata** command in the EEPROM memory.

Reading the IDLOCS:

This code reads the contents of the USER ID in the PIC. In this case the version of this program is sent to the terminal.

User command: **CheckAlarm1Avaible()**

This command allows a user to check if the data of the Alarm1 has been written in the EEPROM memory.

```
$define CheckAlarm1Avaible() GoSub CheckAlarm1Avaible_Sub
```

```
CheckAlarm1Avaible_Sub:
    ALAvaible = ERead EEADR1_Alarm1Avaible
    If ALAvaible = $5A Then
        F_Alarm1Avaible = 1
    Else
        F_Alarm1Avaible = 0
    EndIf
    Return
```

The macro is written with a subroutine because it will be used more than once.

It is the same code for Alarm2.

```
$define CheckAlarm2Avaible()
```

STATE02: (Check the DS1307 acknowledge)

A new global variable has been defined to be used in this module and later.

Dim ReturnACK As Byte

```
DS1307_Present(ReturnACK)
If ReturnACK = 1 Then
    HRSLStrg(TXT14,1)           ' "NACK DS1307 RTC device!"
    NextState(SM_Error_NACK)    ' The program demo cannot run.
Else
    HRSLStrg(TXT13,1)           ' "The DS1307 Device is ready!"
    NextState(SM_Enable1Hz)
EndIf
```

Before proceeding with the program, it is essential to know if the DS1307 circuit is working or connected properly. This function is performed by a command from the library that I wrote some time ago.

There are 2 possible answers.

If the acknowledge received is incorrect, the State Machine is sent to an error module.

NextState(SM_Error_NACK)

If not, it will execute the code in a new module

NextState(SM_Enable1Hz)

STATE03: (Config the DS1307 to Output the 1Hz signal)

```
DS1307_WriteControl(%10010000,ReturnACK)
If ReturnACK = 1 Then
    HRSLStrg(TXT16,1)           ' "NACK DS1307 Write Control"
    NextState(SM_Error_NACK)    ' Go to SM_Error_NACK State.
Else
    AllButtons = 0
    StartTimeout(SM_AskForConfig,10) ' Start Time Out for 10Sec
    NextState(SM_ContrastSetup) ' Configuration of LCD contrast.
EndIf
```

The DS1307 circuit is configured to obtain 1Hz at the corresponding output that is sent to the PIC PORTA.4 to generate an interrupt. After that the SM is sent to a contrast adjustment menu. The contrast of this LCD is adjusted by software.

NextState(SM_ContrastSetup)

SM command: **StartTimeout(ASK_ForConfig,10)**

When entering any menu where a key or a keyboard is read, a timeout is necessary to return to the execution of the main program after some time without typing. The timeout value is 10 seconds in this example. The most interesting thing is that the SM could be sent to any destination as it is not a simple subroutine. The target module would be then "ASK_ForConfig" in case of a timeout after 10 seconds.

SM command: **NextState(SM_ContrastSetup)**

This command sends the SM to a new menu to adjust the contrast.

StateOut()

Depending on the case it is necessary to define the parameters before executing the next code. It is not always possible in the next module and will depend on its construction, especially when there is a loop.

STATE04: (Update Time & Date for the DS1307)

This module is responsible for writing the Date and Time data in the DS1307 circuit. It will be called when the date and time need to be updated. These macros have been updated in the library to introduce the Byte variables as well.

Once the clock update is done, it will go to the main module (SM_ReadTimeDate).

DS1307_WriteDate(VDayOfWeek, VDay, VMonth, VYear, ReturnACK)

DS1307_WriteTime(VHour, VMinute, 0, ReturnACK)

The Minute parameter is always equal to 0 to write.

STATE05: (Running Time & Date)

It is in the main module of the program where everything happens.



Transitional Input State:

The activation time of the BackLight is defined. The TimeOut is deactivated, it is not necessary in this code.

SetBackLightTime(5)

DisableTimeOut()

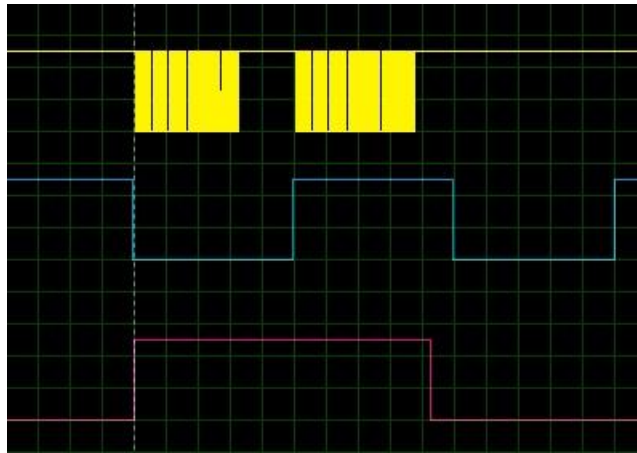
Static State:

Every Second:

The alarms are controlled first. If one alarm is activated then the SM will go directly to the alarm module (SM_Alarm).

The main function of this module is to know the time every second and the date every minute. These 2 functions are synchronized by the 1Hz input from the DS1307 circuit via the PORTA4 pin.

StateInitEnd(SyncOn) ' *Module synchronized with timing 1mS.*



Perfect synchronisation is achieved by reading one second and one minute flags, there is no dead time.

In this case an option of the SM is entered to synchronize the reading of the DS1307 with a timing of 1mS. In this way the interrupt will not affect the I2C communication because an I2C Hardware driver is used. It may be useful for other situations.

The time is reached with the command: **DS1307_ReadTime(ReturnACK)**, always verifying the validity of the I2C communication by verifying "ReturnACK".

To print the results on the LCD and also on the terminal the command is given to the SM2 to run this job after the 1 second interrupt.

NextState2(SM2_PrintTimeDate)

Every Minute:

The activation time of the BackLight is set to 5 seconds: **SetBackLightTime(5)**

The date is reached with the command: **DS1307_ReadDate(ReturnACK)** and again always verifying the validity of the I2C communication with "ReturnACK". Since the code is not in the first line of the module, the command: **Sync1mS()** is used to synchronize the date reading.

A request is done to SM2 to execute a task in the BackGround every minute, in this case to check the activation status of the alarms with "F_DOeveryMinute = **1**".

Next and every second, the time of the BackLight is decremented until deactivated. In this case, to make the LCD slightly visible, a PWM of 30% is set to illuminate the LCD: "PWMBLCounterValue = **3**".

Checking the pushbuttons:

This is a function that we will see in module 8. Each minute the fault is reset to re-check the buttons

```
AllButtonsBitsDefects = %11110000
ButtonFail = 0
```


Reading the pushbuttons:

This device has 4 pushbuttons to perform the main tasks of the calendar. The reading of these pushbuttons cannot be executed at the same time as the date and time readings. It is important not to delay the SM. The reading is chosen at a time when no main task is performed and only once every 10 ms. In this way the tasks are divided over time.

PushButton 1:

This button causes a jump to the LCD contrast adjustment module. A TimeOut of 10 sec is set in case no key is pressed. If the TimeOut is finished, the SM would return to this same destination.

StartTimeOut(SM_ReadTimeDate,10)

To avoid problems with the pressed keys that are difficult to control in a menu, the command is given to check if the button has been raised before going to the destination. (Indirect Addressing)

NextIndState(SM_ButtonsOFF2,SM_ContrastSetup)

PushButton 2:

This button allows you to go to the date/time or alarms configuration module. A TimeOut of 10 sec is set in case no key is pressed. If the TimeOut is finished, the SM returns to this same destination.

PushButton 3:

This button lets you go to the alarm status check module and activate the daily alarm 2.

PushButton 4:

This button allows you to go to this module to disable the alarms.

Transitional Ouput State:

No code

STATE06: (NACK error Module)

A warning of this error is sent to the LCD and the terminal. Any pushbutton may be pressed to interrogate the DS1307 circuit again. If it fails again then the SM would return to this module.

STATE07: (Configuration Time/Date or Alarms)



This menu allows you to choose the way to configure the calendar or alarms. It is a somewhat peculiar system to make the most of SM.

When calling this module you have to define: AllButtons = 0

This value is recognized as reading the keyboard with a TimeOut of 10 seconds.

StartTimeOut(SM_ReadTimeDate,10)

NextIndState(SM_ButtonsOFF,CallerState)

If the TimeOut is activated, it will return to the main module (5). The indirect address command is used to read the pushbuttons.

CallerState

It is the generic information to inform the SM that it must return to this same module after having read a pushbutton like a Gosub command.

STATE08: (Check if all Buttons are OFF)

The system of reading of the pushbuttons or of a keyboard behaves in 3 parts.

- 1- Check if the push buttons are open to follow.
- 2- Read any tight buttons.
- 3- Wait until the push buttons are open to follow.

In this way a reading of a pushbutton or a keyboard will never fail.

Step 1: Check that the push buttons are open to follow.

GoSub ReadButtonsPORT_Sub *' Init & Read All Buttons.*

The code written in a subroutine will be read twice. In this first reading all the inputs of the pushbuttons are configured previously and then the value of all the bits input are read for module SW_ButtonsOFF.

In the second reading in Step 3, the inputs are only read for module SW_ButtonsOFF2. It is a very safe way to read a few inputs.

Then their values are controlled. According to the schematic of the LCD shield all the pushbuttons hold a resistor to VDD. If all the buttons are open the corresponding value will be %11111111 or 255 in decimal.

If ButtonsVirtualPORT = 255 **Then**

NextStateDelay(30,IncState) *' Delay 30mS to exit the state.*

EndIf

It verifies the value of the virtual port containing all the input bits. If all the pushbuttons are up, it goes to the next state with a delay of 30 mS. The other option, that is never contemplated, is to have a broken button stuck. In this case the program would be locked within an infinite loop in the SM.

To avoid this problem, and as an example every SM loop is counted and after passing 5 seconds a fault is generated. For that we need a little delay between readings.

StartDelayMS(30)

It is the code that will be used in 2 different modules.

```

CheckButtonsFail_Sub:
  Inc ButtonsOffCounter
  If ButtonsOffCounter = 167 Then ' <= 167 x 30mS = 5 Seconds
    ButtonsOffCounter = 0
    CLL(2)
    If ButtonsVirtualPORT.0 = 0 Then
      ButtonsFail = 1
    ElseIf ButtonsVirtualPORT.1 = 0 Then
      ButtonsFail = 2
    ElseIf ButtonsVirtualPORT.2 = 0 Then
      ButtonsFail = 3
    ElseIf ButtonsVirtualPORT.3 = 0 Then
      ButtonsFail = 4
    EndIf
    NextState(SM_ReadTimeDate) ' Come back to Main Module.
  EndIf
Return

```



STATE09: (Read all Buttons)

Step 2: Read the pushbuttons

With the help of a counter (**KEYCounter**), a debounce system is built; Counted 4 equal readings to generate a correct answer. Pressing a key loads a specific value into the "AllButtons" variable.

```

If F_SW1 = 0 Then ' Check Button 1.
  AllButtons = 1
  IncState()

```

After reading a key, it goes to the next state.

Reading & checking the Buttons.

```

ReadButtonsPORT_Sub:
  ANSEL = 0 ' PORTA digital
  TRISA = TRISA | %00001111 ' SW0 to F_SW3 for input.
ReadButtons2PORT_Sub:
  If ButtonFail > 0 Then
    SetBit AllButtonsBitsDefects, ButtonFail - 1 'Disable PushButton
  EndIf
  ' All Buttons failed are saved in the AllButtonsBitsDefects buffer.
  ButtonsVirtualPORT = PORTA | AllButtonsBitsDefects 'Read PushButtons
Return

```

SM command: CheckTimeOut()

This SM command checks if the Delay Time Out is finished and loads the new destination for the SM.

STATE10: (Check if all Buttons are OFF2)

Step 3: Wait for all push buttons to be open.

It is the same routine as step 1 with a small difference. Once the key has been lifted and given as valid, the SM is sent to a generic destination.

ReturnIndState(0) *' Load the Return State defined by the caller.*

The SM is sent to the module that requested execute state 8 (Step 1). It could be any program module. Indirect addressing is used. The **0** means that there is not a delay.

The same code controls the fault of the pushbutton (pushbutton always pressed).

STATE11: (Get the value of Day of Week for calendar)



It is a menu to determine the day of the week.

Depending on the key pressed, a specific function is executed, incrementing, decrementing, validating the value or reading the keyboard.

In order to display a value at the beginning of the routine, it is necessary to send some parameters compatible with the format, for example:

AllButtons = 3 *' Like the Button 3 pressed (+)*

VDayOfWeek = 0 *' Initialise parameter.*

StartTimeOut(SM_ReadTimeDate,10) *' Start a Time Out for 10 seconds*

NextState(SM_GetWeek) *' Start configuration.*

And you can read the word "**Sunday**" on the LCD.

When you go to another menu, you have to start a 10-second TimeOut with the command:

StartTimeOut(SM_ReadTimeDate,10) *' Start a Time Out for 10 seconds*

STATE12: (Get the value of Day)

It is exactly the same structure as the previous one for Day.

STATE13: (Get the value of Month)

It is exactly the same structure as the previous one for Month.

STATE14: (Get the value of Year)

It is exactly the same structure as the previous one for Year.

STATE15: (Get the value of Hour)

It is exactly the same structure as the previous one for Hour.

STATE16: (Get the value of Minute)

It is exactly the same structure as the previous one for Minute. As previously determined, the date and time or an alarm will be written.

STATE17: (Contrast Setup)



This is the main menu for adjusting the contrast of the ST7036 LCD. You can see that there is a response different from the TimeOut as it comes from the Reset or the main menu. If the key 1 has been pressed it goes to the setting menu.

```
If F_MenuContrastFromReset = 1 Then
  StartTimeout(SM_AskForConfig,10)  ' Start a Time Out
Else
  StartTimeout(SM_ReadTimeDate,10)  ' Start a Time Out
EndIf
```

STATE18: (Adjust the Contrast)

Adjust the contrast value in real time by writing the value on the LCD to see the contrast variation.



And at the end you press the key 2 and go to another menu to record the value in the EEPROM.

STATE19: (Save the Contrast to eeprom)

In this new menu you can choose whether or not to record the contrast value in the EEPROM.

STATE20: (General Alarm Menu)



This menu allows for enabling alarm 1 or 2 and to go to the corresponding module to set the parameters.

STATE21: (Save Alarm parameters)

This menu allows a user to save the alarm parameters to the EEPROM memory.

STATE22: (Alarm Warning)

When an alarm is triggered from the SM2 in the background it goes directly to this module. To reset this alarm pressing any button is enough. As the alarms are routed twice, a subroutine has been generated.



```
ClearAlarms_Sub:
  AllButtons = 0
  If F_Alarm1 = 1 Then
    F_Alarm1 = 0
    EWrite EEADR1_Alarm1Avaible,[$FF] ' Erase the Alarm1
    F_Alarm1Avaible = 0
    F_EnableAlarm1 = 0
  EndIf
  If F_Alarm2 = 1 Then
    F_Alarm2 = 0
    F_EnableAlarm2 = 0
  EndIf
  StopFunction1()
  Return
```

STATE23: (Display Alarm 2)

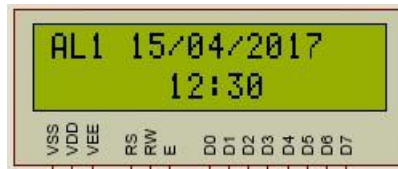
The status of the alarm 2 is displayed and the daily alarm is activated.
It is done by pressing push button number 3 in the main state "SM_ReadTimeDate".



STATE24: (Display Alarm 1)

The status of the alarm 1 is displayed and the general alarm is activated.
It is done by pressing push button number 3 in the main state "SM_ReadTimeDate".





STATE25: (Disable the Alarms)

This menu is called by the pushbutton 4 from the “SM_ReadTimeDate” and allows a user to disable alarms 1 or 2 individually.



STATE MACHINE 2: (Running in the BackGround)

State machine number 2 is activated with a specific command from SM1 and works in the same way.

NextState2(SM2_PrintTimeDate)

STATE101: (Print Time, Date & Alarms on the LCD and The Terminal)

It executes jobs after every second and/or every minute flags when the SM1 orders it.

The Time and Date is printed every second or minute.

Alarms, if activated, are controlled every minute

The error message of a fault is printed if it is necessary.

The programmed alarm indication is printed on the right side of the LCD.



This routine, in SM2, does not delay the State Machine 1 because it is executed in downtime of SM1.

CONCLUSION

I have tried to explain all the tricks that can be done with my state machine to carry out a project. Surely there will be more to discover, it will depend on you and your imagination. I think it is easier to program with this system with projects that allow it, of course.

I wish you all the best in your projects.

Enjoy the State Machine!

State Machine Part4.

Alberto Freixanet

12 April 2017