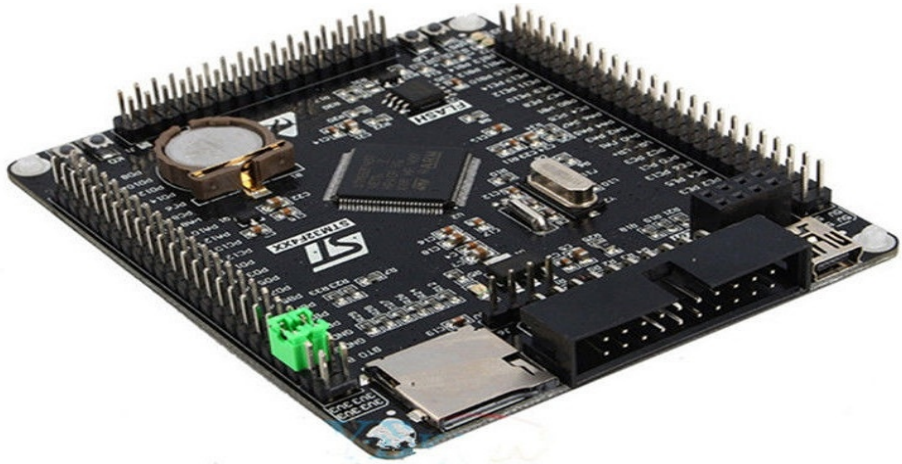
STM32F4_Timer1_InputCaptureISR

Comments.html (C) 2020 Helmut Weber

Board Definition: STM32GENERIC
https://danieleff.github.io/STM32GENERIC/

Board: BLACK F407VE/VG
Specific Board: BLACK F407VE (2.0)
(This includes freeRTOS !)

Image from: https://os.mbed.com/users/hudakz/code/STM32F407VET6_Hello/shortlog/



Microcontroller features

- STM32F407VET6 in LQFP100 package
- ARM®32-bit Cortex®-M4 CPU + FPU
- 168 MHz max CPU frequency
- VDD from 1.8 V to 3.6 V
- 512 KB Flash
- 192+4 Kbytes of SRAM including 64-Kbyte of CCM (core coupled memory) data RAM
- GPIO (82) with external interrupt capability
- 12-bit ADC (3) with 16 channels
- 12-bit DAC (1) with 2 channels
- RTC
- Timers (14)
- I2C (3) interfaces (SMBus/PMBus)
- I2S (2)
- USART (4)
- SPI (3)
- USB 2.0 full-speed
- USB 2.0 OTG
- CAN (2)

Board features

- JTAG/SWD header
- Micro SD
- Winbond W25Q16 16Mbit SPI Flash
- RTC battery CR1220
- 10/100 Ethernet MAC
- 3.3V LDO voltage regulator
- Mini USB connector
- 1x power LED
- 2x user LEDs D2 (PA6) D3 (PA7)
- 2x jumpers for bootloader selection
- Reset button,
- Wakeup button
- 2x user buttons K0 (PE4) and K1 (PE3)
- 2x24 side pins + 2x16 bottom pins + 1x4 ISP pins
- 2x16 FMSC LCD Interface
- NRF24L01 socket

Purpose of this program

1) Build and use a Program-Template to use with different MCUs, Operating Systems, Tasks and Interrupts

2) Test the DA Converter and Timers of the BLACK STM32F407 with and without Interrupts

4) High precision time measurements with Timer Input Capture (6 ns ticks !)

5) Test timings of using and RTOS and CoopOS - minimize gap between Task Switches

6) Show Real Time Capabilities of CoopOS

If you are in a hurry: Conclusion

I Template (II Timer_Input_Capture_DAC)

Template for multitasking MCU projekts, mainly used with Arduino-IDE.

This is a

- Start with Config.h

Here we define the MCU which we want to use (here: _STM32F4_)

Next we define the OS to use: (_RTOS_ or _COOPOS_ or none of them)

Then some #defines for this version of the program are following (_MYSER_)

- Processor specific details are typically found in Defines.h

For instance the access to Port-Bits instead of digitalWrite(xx)

- CoopOS.h, CoopOS_Defines.h and CoopOS.cpp

are included if _COOPOS_ in Config.h is defined. They should not be changed if you

are not shure what the do.

If _COOPOS_ is defined you `main()` or `setup()` must have:

```
#ifdef _COOPOS_

    //TaskInit(char *name, void (*f)(void *), uint8_t priority, uint8_t stat,
```

```
uint32_t del, void *param) ;

#ifdef _DAC_OS_
    TaskInit("DAC-Out     ", vTask4, 100, READY, 0, NULL) ;
#endif

    TaskInit("TaskList    ", vTask2, 100, READY, 0, NULL) ;

#ifdef _MYSER_
#include "MySerial.h"
    extern void MySer_Task(void *);
    TaskInit("MySerial    ", MySer_Task, 100, READY, 0, NULL) ;
#endif


    TaskInit("Blink       ", vTask3, 100, READY, 0, NULL) ;
    TaskInit("TaskSwitches", vTask1, 100, READY, 0, NULL) ;


    while(1) {
        Scheduler();
    }

#endif // _COOPOS_
```

In Arduino-IDE `loop()` must be defined, but it is never called !

loop() is only called, if _COOPOS_ nor _RTOS_ are defined !

- CoopOS_Tasks.h and CoopOS_Tasks.cpp

Here are the tasks which could be used in main() / setup() to with TaskInit().

Here is a typical CoopOS task: CoopOS_Task

CoopOS Tasks begin with `taskBegin()` and end with `taskEnd()`. Without that they will not work!

Local variables which should survive a `taskSwitch()` / `taskDelay()` MUST be defined static !

CoopOS depends on macros. The usable macros are defined in **CoopOS_Defines.h**

- RTOS_Tasks.h, RTOS_Tasks.cpp

Here are the RTOS-Tasks which could be created in main() / setup().

```
    #ifdef _RTOS_
xTaskCreate( vTask1,
               "TaskSwitches",
               configMINIMAL_STACK_SIZE,
               NULL,
               tskIDLE_PRIORITY + 1,
               NULL);
```

```
    xTaskCreate( vTask2,
                 "TaskList    ",
                 configMINIMAL_STACK_SIZE+300,  // <<<<<<<<<<<<<<<<
                 NULL,
                 tskIDLE_PRIORITY + 1,
                 NULL);

    xTaskCreate( vTask3,                    // Blink
                 "Blink       ",
                 configMINIMAL_STACK_SIZE,
                 NULL,
                 tskIDLE_PRIORITY + 1,
                 NULL);

#ifdef _MYSER_
    xTaskCreate( MySer_Task,                // MySer_Task
                 "MySer_Task  ",
                 configMINIMAL_STACK_SIZE,
                 NULL,
                 tskIDLE_PRIORITY + 1,
                 NULL);
#endif


#ifdef _DAC_OS_
    xTaskCreate( vTask4,                    // DAC
                 "DAC-Out     ",
                 configMINIMAL_STACK_SIZE,
                 NULL,
                 tskIDLE_PRIORITY ,
                 NULL);
#endif


          vTaskStartScheduler();            // Never returns so loop() is never
called !

#endif // _RTOS_
```

II Timer , Input Capture, DAC

The STM32F4 has a nice feature: Timers can create (or count) pulses without using th CPU - no interrupts, no program code is used (well, Timers must be initialzed).

Timer3
is used to produce pulses up to 14 MHz.
They are produced at pin PA8

Timer1
is used to count 1/168 ticks per us from one rising edge to the next at the input pin to measure the pulse frequency.
The input pin of Timer1 is set to pin PE9

PA8 and PE9 are connected !

DAC
The STMF407 has a Digital to Analog Converter and I want to test this feature.
The analog output comes at pin PA4 where it can be measured with a scope.
The code for DAC is found in DAC.h and DAC_cpp

Tests without Interrupt

The first test is done with

1) _DAC_BRUTEFORCE_ (measure fastest DAC output)

These warnings are emitted by the compiler to control the settings in Config.h, if _SHOW_DEFINES_ is set:

```
#warning "_DAC_BRUTEFORCE_ defined"
#warning "_COOPOS_ UNdefined"
#warning "_RTOS_ UNdefined
#warning "_DAC_ defined"
#warning "_DAC_ISR_ UNdefined"
#warning "_DAC_OS_ defined"
#warning "_TIMER1_ISR_ UNdefined"
#warning "_MYSER_ UNdefined"
```

No serial output, no digital pins involved. Only analog output at PA4 for the scope!
The code at the end of setup() is running. It shows the fastest DAC conversion possible. No Timers are involved.

I can measure an sample rate of about 11 MHz (91 ns per sample).
BUT: It needs 2 us to come to zero at the change from 255 to 0 !!!
I don't know if this is typical fpr the STM32F407 or my scope with probes - but I think, it is the STM32F4.
The next test:

2) Loop() (Timer1 and Timer3 are used)

```
#warning "_DAC_BRUTEFORCE_ UNdefined"
#warning "_COOPOS_ UNdefined"
#warning "_RTOS_ UNdefined"
#warning "_DAC_ defined"
#warning "_DAC_ISR_ UNdefined"
#warning "_DAC_OS_ defined"
#warning "_TIMER1_ISR_ UNdefined"
#warning "_MYSER_ UNdefined"
```

The loop() is running. It emits DAC at PA4 with highest speed (scope).
Once per second the pulse length at PA6 / PE9 is measured and displayed in 1/168 ticks.

PA6 and PE9 must be connected.

The output:

```
02:30:14.993 ->  16
02:30:14.993 ->  (Loop) 10.50 MHz
02:30:14.993 ->  DA_Conv: 1132943
```

shows the 10,5 MHz pulses are detected exactly and 1.1 MHz DAC samples are

produced in loop()
The 2 us delay from 255 to 0 is to be seen as well.

3) _RTOS_

```
#warning "_DAC_BRUTEFORCE_ UNdefined"
#warning "_COOPOS_ UNdefined"v
#warning "_RTOS_ defined"
#warning "_DAC_ defined"
#warning "_DAC_ISR_ UNdefined"
#warning "_DAC_OS_ defined"
#warning "_TIMER1_ISR_ UNdefined"
#warning "_MYSER_ defined"
```

Now an OS is used: freeRTOS.
The tasks of RTOS_Tasks.h/.cpp are working. These are:

a)
In Reality most RTOS-Tasks are cooperative !!!
Most tasks do have a taskYIELD() or a vTaskDelay(ms) or are waiting for something.
They do not use the full ticktime of 1 ms to get preempted !

```
void vTask1(void *pvParameters) {                              // show task switches LA
    pinMode(PD7,OUTPUT);
    for (;;) {
        GPIO_PIN_SET(GPIOD,7);
        Switches++;
        GPIO_PIN_RESET(GPIOD,7);
        taskYIELD();
    }
}
```

This task is a measurement how often the RTOS is able to do task switches. It runs as often as possible (with lower priority)
and is a replacement for the IDLE Task.

Here we get amazing 1 Million task switches per second !

b)
vTask2() is called once a second and does the measurement of pulses created with Timer3 and counted at the input of
Timer1 (PA6,Input Capture) and displays the statistics.

Attention: The Input Capture Measurement is blocking for one pulse period to do it fast enough.
Here we have 16 ticks of 1/168 us and ~= 1/10 us and this wil do no harm. But have it in mind for lower frequncies!

c)
vTask3() does a simple blink of the builtin LED to show the program is running.

d)
vTask4() does a DA-Convers dision every 1 ms. Due to the 1 ms tick rate of the RTOS we get the maximum of 1000 DAC per second.
We have to use tricks to increase this number !

e)
MySer_Task()

Serial output can be a bottleneck for every embedded system!

First method to increase is to output only short shunks of Serial.print(..)
and put a TaskYIELD() {RTOS} / taskSwitch() {CoopOS}
after it.

The second method is to print into a ringbuffer (MySer.print) and let a task
(MySer_Task) do a timed output to the serial line.
This is done if _MYSER_ is set.

MySer_Task() can be used with RTOS or CoopOS !

Here is the output:

```
12:08:50.149 ->  (DAC: OS)  Diff:16
12:08:50.149 ->  (RTOS) 10,5000 MHz   DA-Conv 1.000
12:08:50.182 ->  T-Switches:990.424
12:08:50.182 -> ****************************************
12:08:50.182 -> Task        State    Prio    Stack    Num
12:08:50.216 -> ****************************************
12:08:50.216 -> TaskList     R        2        306    2
12:08:50.216 -> TaskSwitc    R        1        109    1
12:08:50.249 -> MySer_Tas    R        1        101    4
12:08:50.249 -> IDLE         R        0        119    6
12:08:50.249 -> Blink        B        2        102    3
12:08:50.249 -> DAC-Out      B        2        107    5
12:08:50.249 -> Tmr Svc      S        2        225    7
12:08:50.282 -> ****************************************
```

The 10.5 MHz pulses are detected and measured correctly.
The 1000 DAC per second are done correctly.
There are nearly 1000000 task switches per second (one per microsecond) !

4) _COOPOS_

If we change from _RTOS_ to _COOPOS_ (Config.h) we get nearly the same
functionality.
Exception: DAC-OUT now produces 100000 DAC-Converions per second instead of
1000.
This is possible because (CoopOS)-taskDelay(us) uses microseconds instead of
milliseconds.

Sometimes it may be a reason to use CoopOS !

If you compare CoopOS_Tasks.cpp and RTOS_Tasks.cpp you'll find them very
similar.
And even TaskInit() and xTaskCreate() are looking nearly the same.
In fact it is often no problem to convert tasks from one OS to the other !

And here is the output;

```
 #warning "_DAC_BRUTEFORCE_ UNdefined"
 #warning "_COOPOS_ defined"
 #warning "_RTOS_ UNdefined"
 #warning "_DAC_ defined"
 #warning "_DAC_ISR_ UNdefined"
 #warning "_DAC_OS_ defined"
 #warning "_TIMER1_ISR_ UNdefined"
 #warning "_MYSER_ defined"

7:21:26.669 ->  Diff:16
17:21:26.669 ->  (CoopOs) (DAC: OS)  T3: 10,5000 MHz   DA-Conv 99.766
17:21:26.669 ->  T-Switches 505.465
```

```
17:21:26.669 ->
17:21:26.669 -> ---
17:21:26.669 -> Task            State    Prio    Delay    W-Res     WSig
Last-Called  Times called
17:21:26.669 -> DAC-Out             D    0        9          0        0
225229892    99774
17:21:26.669 -> TaskList            R    1        1009673    0        0
225229892    14
17:21:26.669 -> MySerial            D    2        100        0        0
225229822    9978
17:21:26.669 -> Blink               D    3        50000      0        0
225204047    20
17:21:26.669 -> TaskSwitches        R    4        0          0        0
225229892    395693
17:21:26.669 -> ---
```

We get 100000 DA-Conversion, but only 500000 task switches per second.

Tests with Interrupt

Let's be more realistic. If we have to count pulses on an input pin we should use interrupts.
We attach an interrupt at pin PE9, rising edge.
If Timer3 fires we get an (external) interrupt.
The question is: How many interrupts per second are possible ?
The Interrupt Function External_ISR() has two parts:
Output an analog value: _DAC_
Measure the Pulse Length: _TIMER1_COUNT_

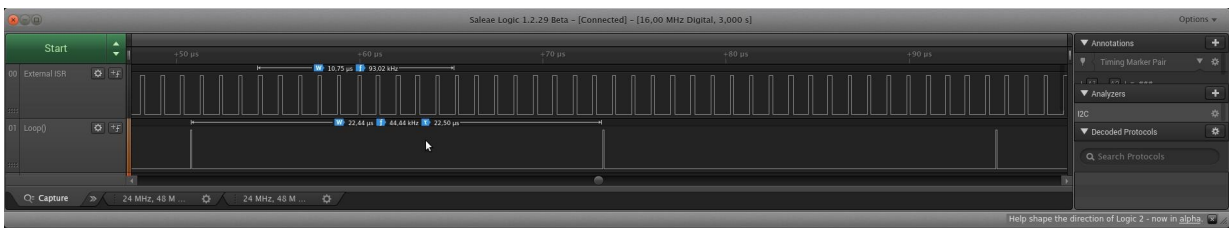
1) Test with bare Interrupt without an OS (=loop()) and without _DAC_ and without _TIMER1_COUNT_ :

```
#warning "_DAC_BRUTEFORCE_ UNdefine
#warning "_COOPOS_ UNdefined"
#warning "_RTOS_ UNdefined"
#warning "_MYSER_ UNdefined"
#warning "_MEASURE_TSWITCH_IN_OUTPUT_ defined"
#warning "_DAC_ UNdefined"
#warning "_DAC_ISR_ UNdefined"
#warning "_DAC_OS_ UNdefined"
#warning "_TIMER1_ISR_ defined"
#warning "_TIMER1_COUNT_ UNdefined"
```

Output:

10:25:56.805 -> 180
10:25:56.805 -> (Loop) (T-ISR) IRQs:933975 0.93 MHz
10:25:56.805 -> DA_Conv: 0

We have exact 933333 Interrupts per second. (Measued with LA):



Also 2000000 Interrupts are possible, but Serial.print(..) will not work any longer ;(

2) Test with bare Interrupt and DAC

```
#warning "_DAC_BRUTEFORCE_ UNdefined"
#warning "_COOPOS_ UNdefined"
#warning "_RTOS_ UNdefined"
#warning "_MYSER_ UNdefined"
#warning "_MEASURE_TSWITCH_IN_OUTPUT_ defined"
#warning "_DAC_ defined"
#warning "_DAC_ISR_ defined"
#warning "_DAC_OS_ UNdefined"
#warning "_TIMER1_ISR_ defined"
#warning "_TIMER1_COUNT_ UNdefined"
```

For this test the ISR-frequency must be lowered a little bit to get serial data out.

Output:

10:30:40.413 -> Diff:200 **<<< This is the time measured from Interrupt to Interrupt in 1 / 168 us ticks**
10:30:40.413 -> (Loop) (T-ISR) IRQs:840723 0.84 MHz
10:30:40.413 -> DA_Conv: 840715

We have exact 840000 Interrupts and DA-Conversions per second. (Measured with LA)

The scope shows a clear and crisp signal.

3) Test with RTOS and Interrupt and IRQ-DAC

But how does it look, if we add an OS (RTOS or CoopOS) ?

Lets test RTOS. We have to lower the frequency again:

What are we doing?

Timer3 generates a frequency of 0.76 MHz. This generate external Interrupts with 0.76 MHz. Timer1 with a resolution of 1/168 us is used to

control the time from interrupt to interrupt: 220 ticks of 1/168 us = every 1.31 us an interrupt.

DA converted output is generated in each interrupt.

If we enable _SHOW_TASKLIST_ (only for information, not for measurements) we see:

```
11:06:30.829 -> ****************************************
11:06:30.829 -> Task         State    Prio    Stack    Num
11:06:30.829 -> ****************************************
11:06:30.829 -> TaskList     R        2        278    2
11:06:30.829 -> TaskSwitc    R        2        107    1
11:06:30.829 -> IDLE         R        0        119    4
11:06:30.829 -> Blink        B        2        102    3
11:06:30.829 -> Tmr Svc      S        2        224    5
```

```
11:06:30.829 -> ******************************************
```

TaskSwitch is a task to count the number of task switches like:

```
while(1) {
   taskYIELD();
   Switches++;
   .

            . measure max time from call to call

   .

}
```

The output is:

```
11:14:00.776 ->  Diff:220
11:14:00.776 ->  (RTOS)  (DAC: T-ISR) 0.76 MHz   DA-Conv 763331
11:14:00.776 ->  T-Switches:108484
11:14:00.776 ->  Delta Task1 us: 63      <<<<<<<<<<<<<<<<<<<<<<<<<<<<  last measurement
11:14:00.776 ->  Max   Task1 us: 567    <<<<<<<<<<<<<<<<<<<<<<<<<<<<  maximum time during all measurements
(maybe since hours)
```

Interrupts and DAC are working well. We get more than 100.000 tasks switches - and average of

a task switch every 10 us. But the are not timed by an interrupt.

RTOS tasks in practice are cooperative ! That means: Very few tasks use the full 1 ms time for their work.
They return to Scheduler with TaskYIELD(), vTaskDelay(), ...
What is the longest gap time we have to take into account between tasks switches.
Why is it importent?
RTOS gives us the garantee to test evers ms which tasks has the highest priority an let it run.
If taskswitches come more often this test is done more often.
Here we have a max time of **567 us** between task switches. This time is due to Serial-print(...) statements.
But often we have embedded systems without any serial output. They show their states using blinking LEDs
or pulling a line up or down or ...
So it is interesting how big the task switch gap is measured outside of serial output.

4) Test with RTOS and Interrupt and IRQ-DAC and buffered Serial Output

```
#warning "_DAC_BRUTEFORCE_ UNdefined"
#warning "_COOPOS_ UNdefined"
#warning "_RTOS_ defined"
#warning "_MYSER_ defined"
#warning "_MEASURE_TSWITCH_IN_OUTPUT_ defined"
#warning "_DAC_ defined"
#warning "_DAC_ISR_ defined"
#warning "_DAC_OS_ UNdefined"
#warning "_TIMER1_ISR_ defined"
#warning "_TIMER1_COUNT_ UNdefined"
```

Enabling _MYSER_ buffers serial output and introduces a task MySer_Task() to do the output to the real world.
The max time between task switches is lowered dramatically:

```
12:18:38.111 ->  Diff:220
12:18:38.111 ->  (RTOS)  (DAC: T-ISR) 0,7636 MHz   DA-Conv 764.047
12:18:38.144 ->  T-Switches:105.843
12:18:38.144 ->  Delta Task1 us: 33      <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  ( from 63 us )
12:18:38.144 ->  Max   Task1 us: 157    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<  ( from 567 us )
```

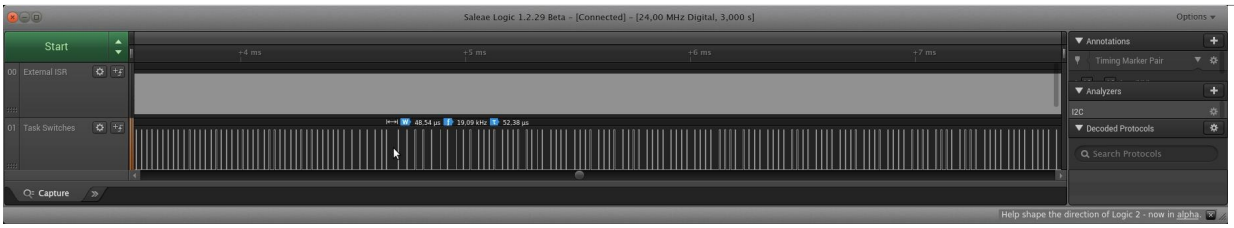
5) Test with RTOS / CoopOS and Interrupt and IRQ-DAC and buffered Serial Output

#warning "_DAC_BRUTEFORCE_ UNdefined"

#warning "_MYSER_ defined"
#warning "_MEASURE_TSWITCH_IN_OUTPUT_ defined"
#warning "_DAC_ defined"
#warning "_DAC_ISR_ defined"
#warning "_DAC_OS_ UNdefined"
#warning "_TIMER1_ISR_ defined"
#warning "_TIMER1_COUNT_ UNdefined"

Comparision:

#warning "_RTOS_ defined"	***#warning "_COOPOS_ defined"***
#warning "_MYSER_ defined"	***#warning "_MYSER_ defined"***
RTOS	**CoopOS**
14:43:36.359 -> Diff:220	15:05:23.011 -> Diff:220
14:43:36.392 -> (RTOS) (DAC: T-ISR) 0,7636 MHz DA-Conv 756.703	15:05:23.011 -> (CoopOs) (DAC: T-ISR) T3: 0,7636 MHz DA-Conv 763.137
14:43:36.458 -> T-Switches:105.804	15:05:23.044 -> T-Switches 50.425
14:43:36.458 -> Delta Task1 us: 9	15:05:23.044 -> Delta Task1 us: 39
14:43:36.491 -> **Max Task1 us: 121**	15:05:23.044 -> Max Task1 us: **54**



CoopOS is better during buffered Serial.print(...)

6) Test with RTOS (AND CoopOS) and Interrupt and IRQ-DAC and buffered Serial Output, but not measured during output

If **_MEASURE_TSWITCH_IN_OUTPUT_** is not defined the max time gap of task switches is NOT measure during **Serial.print (..)**
This is interesting, if we want to estimate values for a system without serial output though the **MySer_Task()** output is included in the measurement !

Comparision:

RTOS	**CoopOS**
14:42:03.163 -> Diff:220	5:07:14.277 -> Diff:220
14:42:03.196 -> (RTOS) (DAC: T-ISR) 0,7636 MHz DA-Conv 756.732	15:07:14.277 -> (CoopOs) (DAC: T-ISR) T3: 0,7636 MHz DA-Conv 763.130
14:42:03.229 -> T-Switches:105.896	15:07:14.310 -> T-Switches 50.645
14:42:03.262 -> Delta Task1 us: 9	15:07:14.310 -> Delta Task1 us: 26
14:42:03.295 -> **Max Task1 us: 83**	15:07:14.310 -> Max Task1 us: **42**

CoopOS is better during *outside* of Serial.print(...)

Conclusion

Timer3 produces pulses with > 750 kHz

They are wired (external) to an Input Pin producing Interrupts on rising Edges: >750000 Interrupts per second

The Interrupt Routine produces a DA-Conversion to build a Sine of 16 Points >750000 times per second (Sine: >48 KHz)

With RTOS and CoopOS some Tasks are running to do and print Measurements (using 6 ns Ticks)

With RTOS and CoopOS one Task (Task1) tries to get scheduled as often as possible (>50000 times per second)

Task1 never has to wait longer than 100 us (RTOS) or 50 us (CoopOS), respectivly.

In this System CoopOS can be seen as a Realtime Operating Sytem which guarantees each task not waiting longer than 50 us for execution time (if READY)

Reasons to use CoopOS

Small MCU with limited memory

No RTOS available

Highest Portability (no porting to a new processor neccessary)

Easy to change for your needs - very small source code (2 pages printout)

Easy to learn - though Semaphores and Signals (even from Interrupt)s are included

taskDelay(us) in microseconds - not milliseconds

Very easy to convert sources to any RTOS

Use of existing (Arduino-) Libraries.RTOSs often have to use new ones because of preemption

Best Jobs for CoopOS:

- **Small MCUs**
- **Very short tasks / Parts of a Task**
- **Very short Delay-Times**
- **Rapid Prototyping with existing and tested Libraries**
- **Fast Test of a new CPU**

Reasons to use Arduino-IDE

Easy to use with most Libraries / Help in Internet

Programs easy to port to another MCU

Program abstraction (digitalWrite, attachInterrupt())

Using low Level MCU Registers possible (replace digitalWrite())

Clear defined Toolchains

Examples

Config.h (Example)

```
// select one processor

// ----------------------------------------


                        #define _STM32F4_
                        //#define _STM32F1_

                        // Rules
                        #ifdef _STM32F4_
                        //#include "stm32f4xx.h"
                        #endif


                        // Rules: Only one MCU is allowed

                            #ifdef _STM32F1_
                              #undef _STM32F4_
                              #include "stm32f1xx.h"
                            #endif

                  #ifdef _STM32F4_
                    #undef _STM32F1_
                  #endif


   // select one or non OS (No OS AND No TIMER1_ISR: ==Loop):

   // use RTOS-Tasks
              //#define _RTOS_

   // use COOPOS-Tasks
              #define _COOPOS_

   // Rules: Make sure that only one or none OS is defined

              #ifdef _COOPOS_
                #undef _RTOS_
              #endif

              #ifdef _RTOS_
               #undef _COOPOS_
              #endif
```

Defines.h (Example)

```
      #include "Config.h"

      #ifdef _STM32F4_
        #define GPIO_PIN_SET(PORT,pin)          ((PORT)->BSRR = 1 <<
(uint32_t)pin)
        #define GPIO_PIN_RESET(PORT,pin)        ((PORT)->BSRR = (1 <<
(uint32_t)pin) << 16)
        #define GPIO_HISPEED_SET(port,pin)      (port)->OSPEEDR = (port)-
>OSPEEDR | (0b11 << (2*pin))
      #endif
```

back_defines

Typical CoopOS Task:

```
void vTask1(void *pvParameters) {                    // show task switches LA//
all local variables are defined here and// MUST be static !!!
             taskBegin();                                  // MUST be there
             pinMode(PD7,OUTPUT);                          // Define pin for
    that
             for (;;) {
                GPIO_PIN_SET(GPIOD,7);
                Switches++;
                GPIO_PIN_RESET(GPIOD,7);
                taskSwitch();                               // back to
    Scheduler        //   OR        //   taskDelay(microseconds);
             }
             taskEnd();                                    // MUST be there
     }
```

back_CoopOS_Task

End of Text