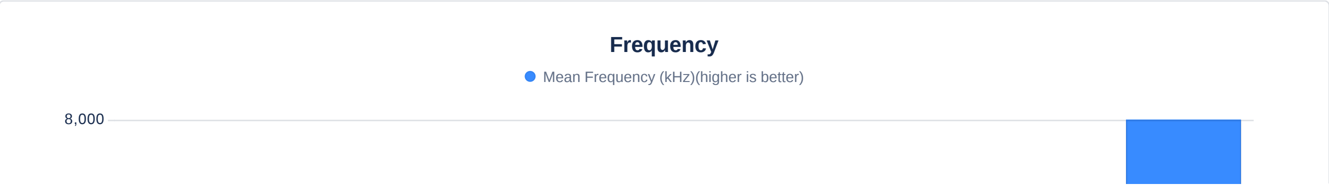
⏱️ Performance

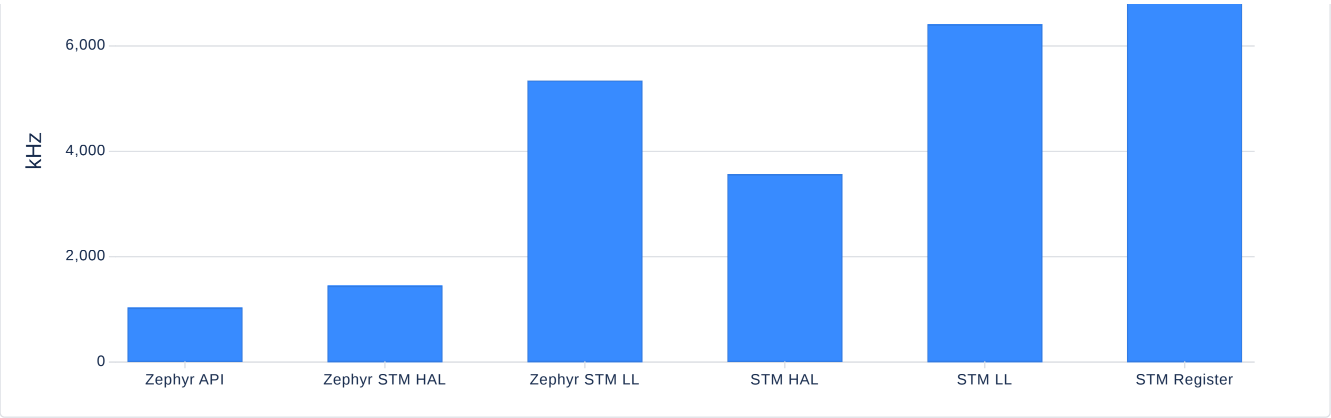
Benchmarking with NUCELO_G071RB.

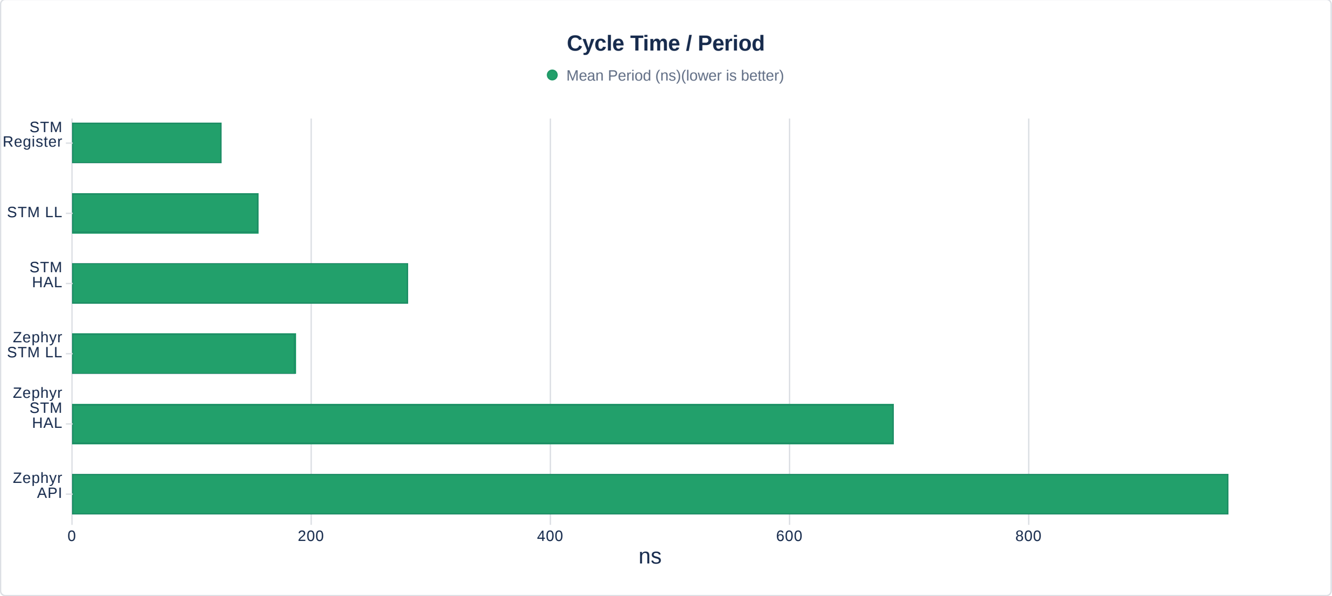
In all tests, the SoC is clocked at 64MHz, and with compiler speed optimisations (-O2).

Results

Test	**Mean Frequency (kHz)** (higher is better)	**Mean Period (ns)** (lower is better)
Zephyr API	1034	967
Zephyr STM HAL	1457	687
Zephyr STM LL	5342	187
STM HAL	3561	281
STM LL	6418	156
STM Register	8011	125







Zephyr

In this test we will use the Zephyr API that allows for hardware agnostic code. `gpio_pin_toggle_dt` does not define for us if any STM code will run, this is only known once we target the compiler to the board, in addition to that, we believe that Zephyr will take preference for the most performance option, at the time of devising this test, I understood this to mean LL over HAL.

Zephyr by default comes with lots of features such as memory protection unit, debug functions, shells, power management etc. To help make these test more comparable to STM tests, these have been turned off and speed optimisations in the compiler have been enabled. These changes didn't appear to make a significant differences in this test, but would have reduced the binary output - a dedicate report into these options would be ideal, rather than to bring all those combinations into this report (there are many combinations).
Additional note, turning off multi-threading yielded a worse result in this test, so that is one zephyr extra that was left enabled. Full list will be available in the source code, in `prj.conf`.

```c
#include <zephyr/kernel.h>
#include <zephyr/device.h>
#include <zephyr/drivers/gpio.h>

// For Zephyr API
#define ZEPHYR_USER_NODE DT_PATH(zephyr_user)
const struct gpio_dt_spec signal = GPIO_DT_SPEC_GET(ZEPHYR_USER_NODE, signal_gpios);

// For STM HAL/LL
#include <soc.h>
// #include <stm32g0xx_hal.h>
```

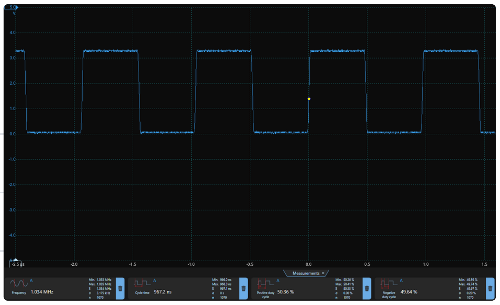
```c
#include <stm32_ll_gpio.h>

// For STM HAL/LL
#define SIGNAL_GPIO_Port    GPIOA
#define SIGNAL_Pin          GPIO_PIN_8

int main(void)
{
    /* Configure the pin */
    gpio_pin_configure_dt(&signal, GPIO_OUTPUT_INACTIVE);
    uint8_t state = 0;
    while (1)
    {
        state ^= 1;

        // ZEPHYR API
        // gpio_pin_toggle_dt(&signal);
        // gpio_pin_set_dt(&signal, state);

        // STM HAL
        // HAL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
        // HAL_GPIO_WritePin(SIGNAL_GPIO_Port, SIGNAL_Pin, state);

        // STM LL
        // LL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
        // LL_GPIO_WriteOutputPort(SIGNAL_GPIO_Port, state<<8);
    }
    return 0;
}
```

Zephyr API

Zephyr API Toggle Test

```c
gpio_pin_toggle_dt(&signal);
```

Mean Frequency (kHz)	1034
Mean Cycle time (ns)	967

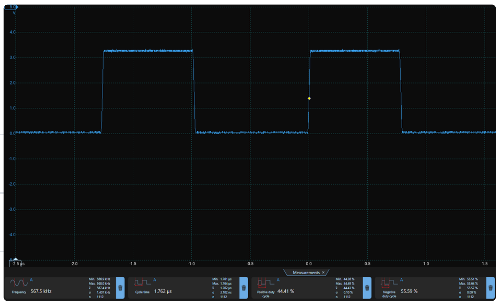


Zephyr API Toggle Test

Zephyr API Write Test

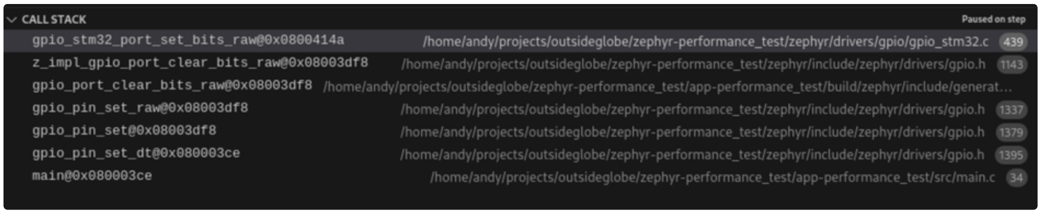
```c
gpio_pin_set_dt(&signal, state);
```

Mean Frequency (kHz)	567
Mean Cycle time (ns)	1762

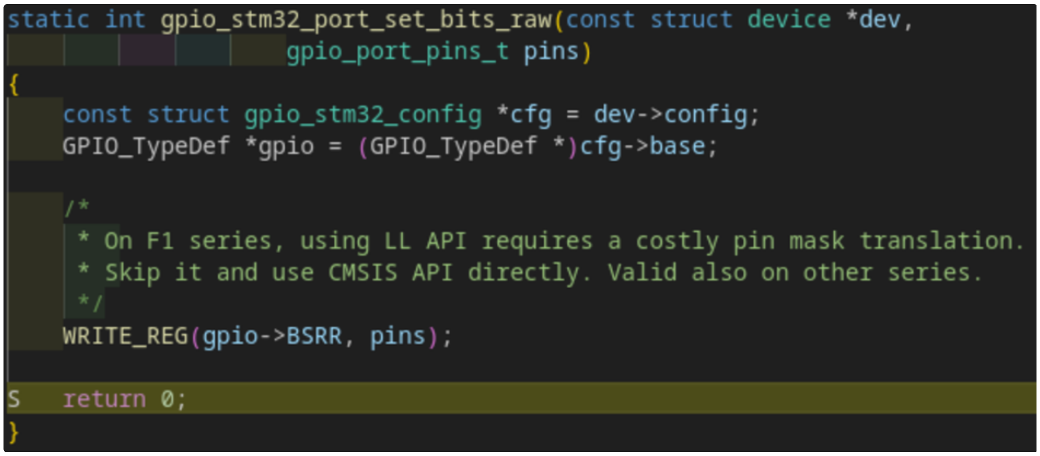


Zephyr API Write Test

Looking into the call stack, and the final function, it becomes clear that in this test, Zephyr is using direct register access via CMSIS, and in the code, it is noted that LL function incurs costly translations.



```
CALL STACK                                                                                                                      Paused on step
  gpio_stm32_port_set_bits_raw@0x0800414a        /home/andy/projects/outsideglobe/zephyr-performance_test/zephyr/drivers/gpio/gpio_stm32.c 439
  z_impl_gpio_port_clear_bits_raw@0x08003df8     /home/andy/projects/outsideglobe/zephyr-performance_test/zephyr/include/zephyr/drivers/gpio.h 1143
  gpio_port_clear_bits_raw@0x08003df8 /home/andy/projects/outsideglobe/zephyr-performance_test/app-performance_test/build/zephyr/include/generat…
  gpio_pin_set_raw@0x08003df8                    /home/andy/projects/outsideglobe/zephyr-performance_test/zephyr/include/zephyr/drivers/gpio.h 1337
  gpio_pin_set@0x08003df8                        /home/andy/projects/outsideglobe/zephyr-performance_test/zephyr/include/zephyr/drivers/gpio.h 1379
  gpio_pin_set_dt@0x080003ce                     /home/andy/projects/outsideglobe/zephyr-performance_test/zephyr/include/zephyr/drivers/gpio.h 1395
  main@0x080003ce                                  /home/andy/projects/outsideglobe/zephyr-performance_test/app-performance_test/src/main.c 34
```



```c
static int gpio_stm32_port_set_bits_raw(const struct device *dev,
                                        gpio_port_pins_t pins)
{
    const struct gpio_stm32_config *cfg = dev->config;
    GPIO_TypeDef *gpio = (GPIO_TypeDef *)cfg->base;

    /*
     * On F1 series, using LL API requires a costly pin mask translation.
     * Skip it and use CMSIS API directly. Valid also on other series.
     */
    WRITE_REG(gpio->BSRR, pins);

S   return 0;
}
```

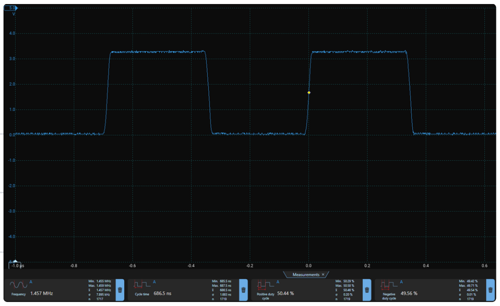
Zephyr STM HAL

We can skip Zephyr API and target directly the HAL. GitHub - zephyrproject-rtos/hal_stm32

Zephyr STM HAL Toggle Test

```
HAL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
```

Mean Frequency (kHz)	1457
Mean Cycle time (ns)	687

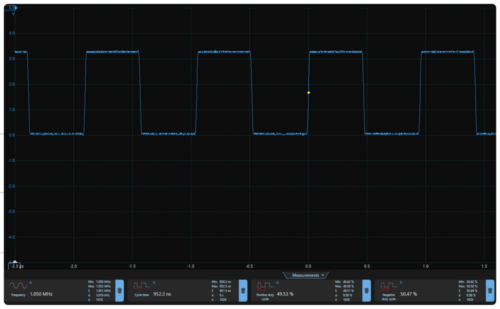


Zephyr STM HAL Toggle Test

Zephyr STM HAL Write Test

```
HAL_GPIO_WritePin(SIGNAL_GPIO_Port, SIGNAL_Pin, state
```

Mean Frequency (kHz)	1051
Mean Cycle time (ns)	952



Zephyr STM HAL Write Test

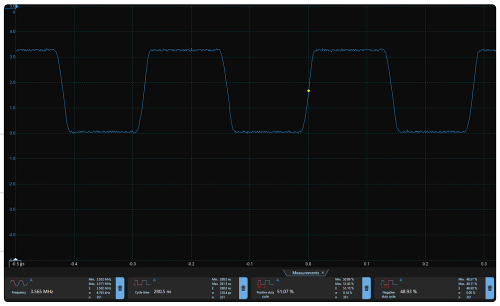
Zephyr STM LL

We can skip Zephyr API and target directly the LL. GitHub - zephyrproject-rtos/hal_stm32

Zephyr STM LL Toggle Test

```
LL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
```

Mean Frequency (kHz)	3562
Mean Cycle time (ns)	281



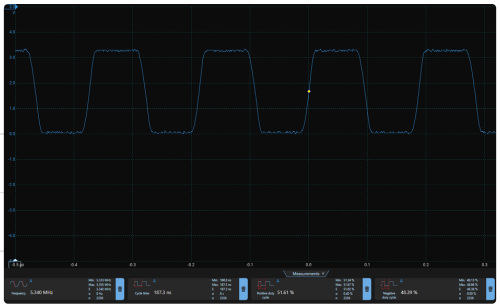
Frequency A 3.565 MHz — Min. 3.552 MHz, Max. 3.571 MHz, x̄ 3.562 MHz, σ 6.783 kHz, n 351
Cycle time A 280.5 ns — Min. 280.0 ns, Max. 281.5 ns, x̄ 280.8 ns, σ 318.4 ps, n 351
Positive duty cycle A 51.07 % — Min. 50.89 %, Max. 51.43 %, x̄ 51.10 %, σ 0.14 %, n 351
Negative duty cycle A 48.93 % — Min. 48.57 %, Max. 49.11 %, x̄ 48.90 %, σ 0.05 %, n 351

Zephyr STM LL Toggle Test

Zephyr STM LL Write Test

```
LL_GPIO_WriteOutputPort(SIGNAL_GPIO_Port, state<<8);
```

Mean Frequency (kHz)	5342
Mean Cycle time (ns)	187



Frequency A 5.340 MHz — Min. 5.333 MHz, Max. 5.355 MHz, x̄ 5.342 MHz, σ 0 Hz, n 2356
Cycle time A 187.3 ns — Min. 186.8 ns, Max. 187.5 ns, x̄ 187.2 ns, σ 0 s, n 2356
Positive duty cycle A 51.61 % — Min. 51.34 %, Max. 51.87 %, x̄ 51.62 %, σ 0.40 %, n 2356
Negative duty cycle A 48.39 % — Min. 48.13 %, Max. 48.66 %, x̄ 48.38 %, σ 0.00 %, n 2356

Zephyr STM LL Write Test

Zephyr (Register)

- □ Research and implement test

STM

```
uint8_t state = 0;

while (1)
{
  state ^= 1;

  // LL
  // LL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
  // LL_GPIO_WriteOutputPort(SIGNAL_GPIO_Port, state<<8);

  // HL
  // HAL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
  // HAL_GPIO_WritePin(SIGNAL_GPIO_Port, SIGNAL_Pin, state);

  // REG
  // GPIOA->ODR ^= SIGNAL_Pin;
  // if (state)
  // {
    // GPIOA->BSRR = (1<<8);
  // }
  // else
  // {
    // GPIOA->BSRR = (1<<8)<<16;
  // }
}
```

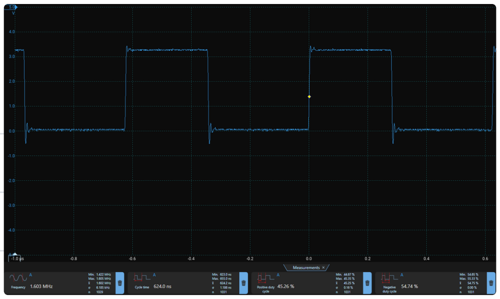
STM HAL

Here we configure the IOC to use HAL for GPIO, this is done in CubeMX after saving and re-generating the code. We then use the HAL_* library function

STM HAL Toggle Test

```
HAL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
```

Mean Frequency (kHz)	1602
Mean Cycle time (ns)	624

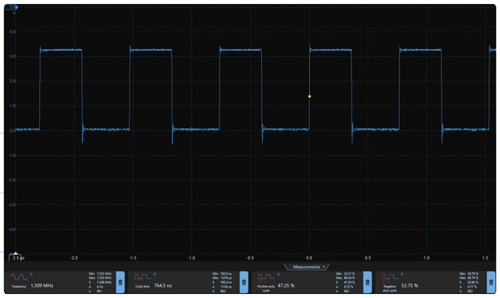


STM HAL Toggle Test

STM HAL Write Test

```
HAL_GPIO_WritePin(SIGNAL_GPIO_Port, SIGNAL_Pin, state
```

Mean Frequency (kHz)	1308
Mean Cycle time (ns)	765



STM HAL Write Test

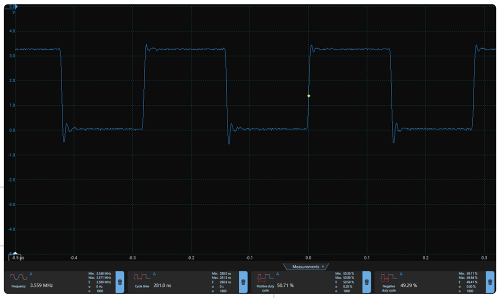
STM LL

Here we configure the IOC to use LL for GPIO, this is done in CubeMX after saving and re-generating the code. We then use the LL_* library function.

STM LL Toggle Test

```
LL_GPIO_TogglePin(SIGNAL_GPIO_Port,SIGNAL_Pin);
```

Mean Frequency (kHz)	3561
Mean Cycle time (ns)	281

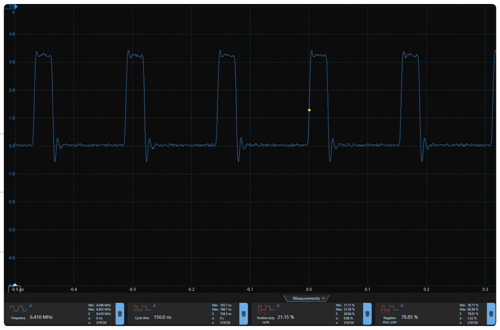


STM LL Toggle Test

STM LL Write Test

```
LL_GPIO_WriteOutputPort(SIGNAL_GPIO_Port, state<<8);
```

Mean Frequency (kHz)	6418
Mean Cycle time (ns)	156



STM LL Write Test

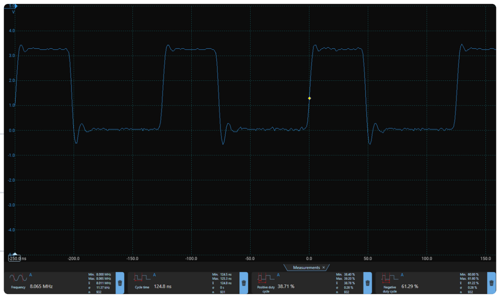
The duty cycle is 80% negative.

STM Register

STM Register ODR Test

```
GPIOA->ODR ^= SIGNAL_Pin;
```

Mean Frequency (kHz)	8011
Mean Cycle time (ns)	125

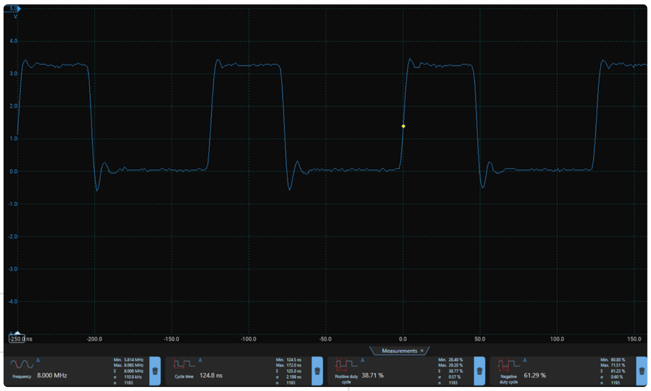


STM Register ODR Test

STM Register BSRR Test

```
if (state)
{
  GPIOA->BSRR = (1<<8);
}
else
{
  GPIOA->BSRR = (1<<8)<<16;
}
```

Mean Frequency (kHz)	8006
Mean Cycle time (ns)	125



STM Register BSRR Test

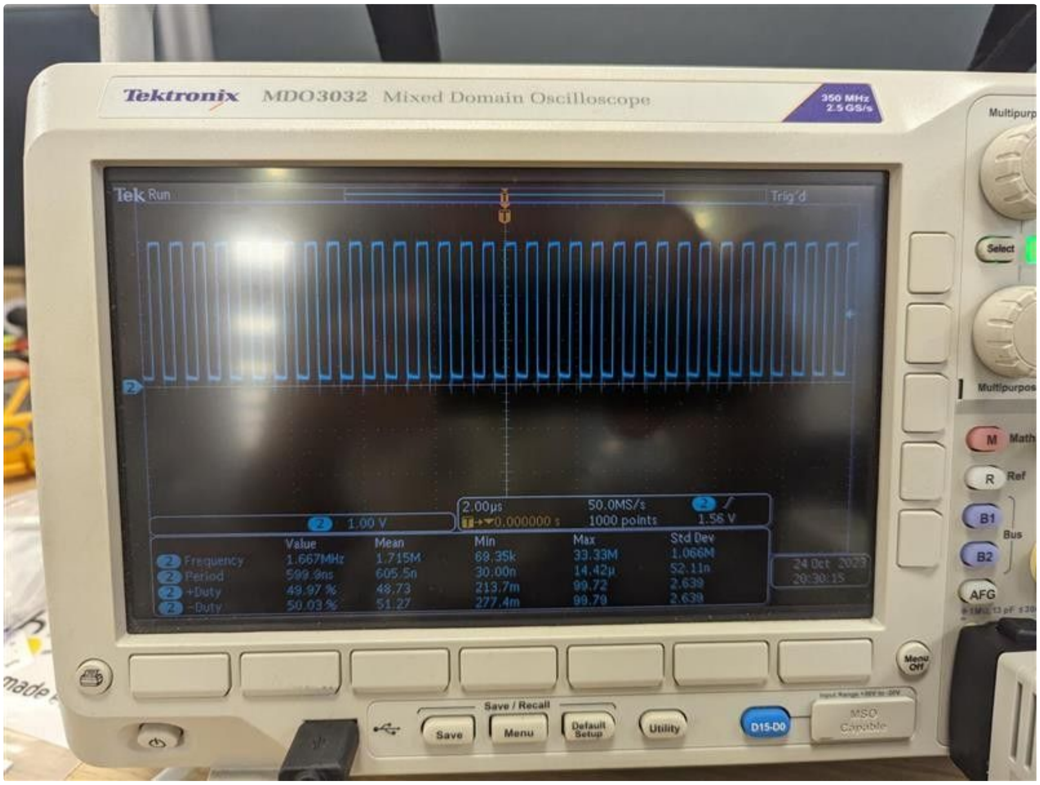
Observations

1. **Zephyr** is out of the box more performance orientated than **STM IDE**, more tuned by default for performance. Out of the box, **Zephyr** (1034kHz) can toggle that GPIO line significantly faster than '**STM LL**' (604kHz), **71% faster.**
2. However, **Zephyr** is using Register banging (via CMSIS) to achieve this performance. So to compare it against register banging '**Bare Metal'** (1526kHz) is **47% faster** than **Zephyr** (1034kHz).
3. Interestingly, using the toggle function was as expected slower for the **STM** tests, but for the **Zephyr** tests it was considerably faster. We saw this again with '**Zephyr STM HAL**' tests, and then again we saw this in the '**STM HAL**' tests once we enabled speed optimisations, showing that Toggle was faster than Write.

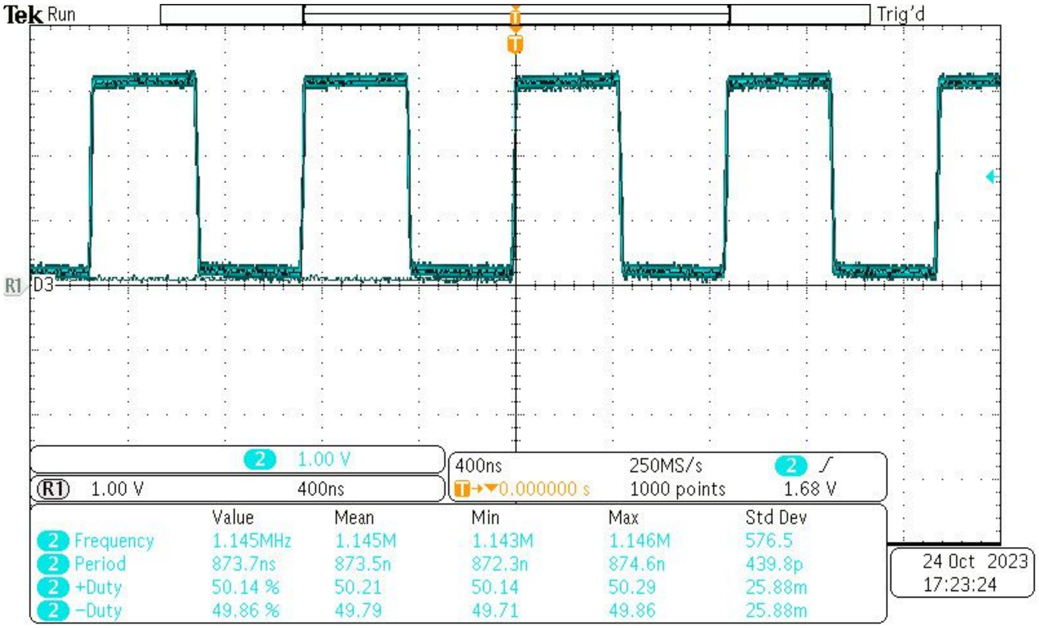
4. ‘**STM LL**’ (6418kHz) is **80% faster** than ‘**STM HAL**’ (3561kHz), but before speed optimisations it was only 7.8% faster.
5. ‘**STM LL**’ (6418kHz) is **20% faster** than ‘**Zephyr STM LL**’ (5342kHz).
6. ‘**STM LL**’ with speed optimisations shows that it takes longer to set than it does to clear.
7. Is **Zephyr** producing a cleaner, squarer wave than **STM**? There are consistent artifacts in the **STM** tests that are not present in the **Zephyr** tests.
8. Speed optimisations in **STM** makes a significant difference, 604kHz to 6409kHz for '**STM LL Write test'**.

old notes:

```
while (1)
{
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
  if (!set)
  {
    GPIOA->BSRR |= (1<<8); // Set the Pin PA5
    set=1;
  }
  else
  {
    GPIOA->BSRR |= (1<<8) <<16;  // Clear the Pin PA5
    set=0;
  }
}
```



	Value	Mean	Min	Max	Std Dev
2 Frequency	1.667MHz	1.715M	68.35k	33.33M	1.066M
2 Period	599.9ns	605.5n	30.00n	14.42µ	52.11n
2 +Duty	49.97 %	48.73	213.7m	99.72	2.639
2 -Duty	50.03 %	51.27	277.4m	99.79	2.639



```
        2   1.00 V                      400ns        250MS/s       2  ⁄
(R1)  1.00 V          400ns             T→▾0.000000 s  1000 points  1.68 V

                Value      Mean      Min       Max       Std Dev
2 Frequency     1.145MHz   1.145M    1.143M    1.146M    576.5
2 Period        873.7ns    873.5n    872.3n    874.6n    439.8p
2 +Duty         50.14 %    50.21     50.14     50.29     25.88m
2 −Duty         49.86 %    49.79     49.71     49.86     25.88m
```

24 Oct 2023
17:23:24