MIDDS

Monitoring Interface of Digital and Differential Signals

# Overview

The **MIDDS** (hereinafter called the Monitor) is a peripheral board to connect to a computer via Ethernet that adds GPIO capabilities to your software. The Monitor does not only work as a reliable General-Purpose Input/Output but also offers high accuracy timestamping. It accepts both TTL and LVDS signals, in the 0V-5V range, and it has a dedicated input to feed your own SYNC signal to synchronize an external clock source with the Monitor.

The MIDDS solves two problems on the same device:

* For one, it works as a digital oscilloscope of high timing accuracy relative to its price.
* And it works as a GPIO card, allowing the computer to generate both input and output signals.

It can therefore be used in the following applications:

* Cost-effective laboratory equipment and instrumentation.
* Sensor calibration.
* Multi-protocol interface adapter (SPI, I2C).
* Board to board communication.
* GPS.
* Test and measurement.

# Capabilities

The Monitor takes advantage of the following STM32H753ZIT6 MCU’s features:

* **High Resolution Timer (HRTIM):** it gives accurate measures up to the frequency of the actual clock signal that is feeding the MCU. The desired running frequency for the CPU will be 400 MHz, generated with an external OCXO of 25MHz, 10 ppb.
* **General Purpose Timers (TIMx):** very similar to the HRTIM but working on half the MCU’s clock frequency (200 MHz).
* **Direct Memory Access (DMA):** this will streamline data exchange between the hardware timer and the software running on the MCU, meaning that the time interval between external input and outgoing message to the PC will be close to negligible.
* **Ethernet compliant with IEEE-802.3-2002:** the MCU will stablish connection with the PC via TCP with a bandwidth of 10 MB/s with a simple and fast custom protocol.
* **GPIOs:** the MCU offers up to 112 GPIOs.
  + In this first iteration of the Monitor, \_\_\_ GPIOs are expected to be used:
  + Both LVDS and TTL input/outputs can be selectively timestamped with different accuracies: high, medium and low.

With all these features, below are listed the expected measuring characteristics of the MIDDS:

* **Guaranteed less than 5 ns of deviation** between real and measured timestamps in medium and high accuracy inputs.
* Clock deviation after SYNC signal: worst-case scenario of 10 ns/s, 600 ns/min, 36 us/h, 864 us/day. It is therefore recommended to use a SYNC signal of a least 1 Hz for high precision measurements.
* **Maximum input frequency of 10MHz.**
* Guaranteed **exact readings of simultaneous clock signals** if they share the GPIO port. If not, a maximum of 10 ns will be allowed between readings.
* Recording of **both rising and falling edges**.
* **Overvoltage protection** on all inputs.
* **High impedance input**, meaning that the Monitor will not affect the input signals.
* **Hardware masked timestamped** signals.

# Working principle

The MIDDS heavily relies on the hardware timers of the MCU. A timer is basically a 16-bit counter, a module within the chip that increments its value by one at the same rate of the MCU’s clock, in case of the HRTIM; and at half its rate, in the case of TIMx.

The counter’s numerical range is somewhat limited (from 0 to 65535). Although this at first may not appear to be a small number, the MCU’s clock is expected to run at around 400MHz, meaning that the counter will fill up every 163.84 microseconds!

When the counter reaches its upper threshold, an overflow occurs, and the counter rolls back to zero. Every time this happens, an interruption or ISR is triggered on the MCU. On this ISR, the MCU increments a software 64-bit counter.

Henceforth, the hardware timers (HRTIM and TIMx) will be called the “fine counters” whilst the software counters will be the “coarse counters”.

A close-up of a sign

Description automatically generated

Figure 1. Counter structure in the MCU.

This structure gives a precision-per-bit of 2.5ns (at 400MHz) and a total time count of 3.022 · 1015 seconds (around 96 million years).

Although this number is impressive, it is not very useful to use such large times without keeping a more constraint time of reference. This is because of the relative accuracy of the onboard OCXO of the MCU: by using an OCXO of 25MHz, 10 ppb, for each pulse of a real 25MHz clock, there is a small amount of time which, at maximum, is being lost:

Therefore, every 25 million pulses, there is a maximum drift of ±10ns. This could amount to 10 ns/s, 600 ns/min, 36 us/h or 864 us/day, which for some applications may be unsuited.

This is why the MIDDS also has a SYNC input that allows the user to input its reference pulse signal. This signal will normally come from a more stable and reliable source, such as GPS. The frequency and duty cycle of said pulse signal will need to be specified via software. Unless noted from the computer, the MIDDS will suppose that a 1 PPS signal of 50% duty cycle is being used.

The following is the synchronization sequence of the SYNC signal with the MIPPS clock:

1. During power up, the OCXO will start heating up, and after 5 minutes, it will generate a stable clock signal of ±10 ppb.
2. When the MIPPS detects that a stable temperature has been achieved on the OCXO it will power up, sending a “READY” message to the computer. It will set its initial time and it will start timestamping from that moment forward in reference to .
3. The SYNC signal is used to calculate the exact time drift the MIPPS internal timers are having in relation with the SYNC signal. This synchronization will happen on both rising and falling edges of the SYNC signal.
4. If the computer is synchronized with the SYNC signal it may also set the current time of the MIPPS in reference to the last SYNC pulse.

# Hardware and Software design

In this section, an overview of the hardware and software design is shown.

## TTL-GPIOs

In Figure 1 a simple diagram of a GPIO is shown.

A diagram of a system

Description automatically generated

Figure 2. Diagram of a single TTL-GPIO pin.

As it can be seen, there is an (almost) direct connection between the external connection of the IO and the GPIO of the MCU, allowing for very fast electronic transitions. The “Voltage Protection” block is an assortment of protection diodes in parallel with the lines that guards against ESD or overvoltage peaks.

Connected to the MCU’s GPIO is an “Edge detector” which basically generates a short pulse every time there is a voltage change on the physical GPIO line, be it working as input or output. This block also guards against spurious edges and environmental noise. Note that the detector also has an “ENABLE” input: this allows the MCU to mask those signals which do not need to be timestamped.

The pulse of the “Edge detector” is fed to one of the seven available HRTIM inputs: on a rising edge, the HRTIM saves its current time value and calls DMA to transport it to a safe place in memory. As all this is hardware controlled, this transaction is very fast. This allows the HRTIM to be “blocked” for very short periods of time, allowing for very fast and close-together reads.

Note that the triggering of the HRTIM happens all on a hardware level. That means that even outgoing signals from the MCU (if the GPIO were to be configured as an output) get timestamped not on receiving the software call, but on the real physical line. This gets rid of delays caused by intermediary interruption calls or slow rising signals (caused by low input impedance on the external device); in other words, the monitoring is always being done on the physical layer, never on the software layer.

After a DMA stores data in memory, an interrupt (ISR) is requested. If the interrupt gets served, the MCU reads the GPIO port associated with the input which caused the HRTIM to be activated. A GPIO port controls the state of 16 individual GPIOs; therefore, when a GPIO port is read, it is being read the state of 16 GPIOs in one go. The MCU then compares the current port read with a latter one, those bits which have changed its value are the ones associated with the GPIO which triggered the ISR.

If the ISR does not get served, due to another more preemptive task, the timer will not be lost. There will be just a small delay in the communications.

Note: what will be most desirable would be to get the HRTIM to trigger two DMA requests: one for the transfer of the timer value from the HRTIM and another for the transfer of the GPIO input register GPIOx\_IDR to memory, but that seems to not be possible without an intermediary software interruption.

## LVDS Inputs

A diagram of a software system

Description automatically generated

Figure 3. Diagram of a single LVDS input.

The LVDS inputs are similar to their TTL counterparts, it only differs by a “Differential to TTL” converter that is put before the GPIO of the MCU. This converter takes the ±400mV input and converts it into a +3.3V TTL signal and vice versa (this is controlled with the DIRection input). Although not pointed out in the diagram, bear in mind that a LVDS signal needs of two lines to work. Both “Voltage protection” and “Differential to TTL” blocks work on two lines. The TTL signal will be generated referenced to common GND.

## Characteristics of timestamped Inputs/Outputs

The MIPPS has three different types of timestamped I/Os:

* High accuracy, with a precision of 2.5ns.
* Medium accuracy, with a precision of 5ns.
* Low accuracy, with a precision of no more than 50ns, although this may depend on the software load of the MCU.

### High accuracy timestamped I/Os

These I/Os are managed by the HRTIM. The HRTIM has five inner timers which are all fed by the general counter of the HRTIM, that is, all five timers get incremented (if enabled) at the same rate as the general counter. In the case of the MIPPS, all those inner timers will always be enabled, meaning that they will always have the same value as the general counter. Moreover, they have been configured to all be reset at the same time the general counter overflows, guaranteeing the same frame of reference for all timers.

As the HRTIM works at the same rate of the MCU’s clock (400 MHz), each increment of these “fine counters” will be 2.5ns.

Each one of those five individual timers can be triggered by a rising or falling edge. When an edge is detected, the value of the timer is transferred through DMA to memory and when that process ends, an interrupt gets called to process that current timer value.

If the MCU could not attend that interrupt request and another edge came, the timer would be overwritten with the time value of the newly arrived edge. That is why there is a maximum input frequency of 1 MHz on all inputs, so that no overwriting can occur.

### Medium accuracy timestamped I/Os

These I/Os are managed by different TIMx of the MCU (TIM1, TIM2…). These timers all have their own independent general counter: some are 16-bit long, whilst others are 32-bit; some can handle four inputs, whilst others can only handle two or even none.

In conjunction, they will all work as 16-bit timers at half the MCU’s clock speed. They will all be reset at the same time by a master TIMx, and this master TIMx will also be reset when HRTIM takes two cycles; therefore, all timers of the MCU are synchronized and the only difference between the high and medium accuracy timestamped I/Os will be the grain of the read but not the frame of reference.

### Low accuracy timestamped I/Os

The remaining GPIOs of the MCU can also be software timestamped. When an edge is detected on one of these GPIOs, an ISR gets triggered. During this ISR, the MCU will grab the current time from the HRTIM and will then timestamp this new edge. Therefore, the accuracy of these I/Os relies on how “free” the MCU is at the moment: if the MCU is not working at that instant, the I/O will be timestamped almost immediately; but if the MCU is doing some other work, this GPIO will be timestamped when it gets free of the more pressing chores, such as the high and medium accuracy I/Os.

## Software priorities

## I/O Organization

In Table 1 it is presented the organization of the GPIOs of the MCU and their respective pin name in the Monitor board. All MCU pins which are used as IO for the Monitor will be **bolded**: differential pins will be called ***DIF\_xn***, and digitals ***DIO\_xn***, where x is the port’s letter of the GPIO and n is a number starting from zero.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PORT A (2xDIFF, 4xDIO)** | |  | **PORT B (2xDIFF, 4xDIO)** | |
| PA0 | Ethernet (ETH\_CRS) |  | PB0 | Ethernet (ETH\_RXD2) |
| PA1 | Ethernet (ETH\_REF\_CLK) |  | PB1 | Ethernet (ETH\_RXD3) |
| PA2 | Ethernet (ETH\_MDIO) |  | PB2 |  |
| PA3 | Ethernet (ETH\_COL) |  | PB3 | Debugging (TSWO) |
| PA4 |  |  | PB4 |  |
| PA5 |  |  | PB5 | HRTIM\_E (HRTIM\_EEV7) |
| PA6 |  |  | PB6 | HRTIM\_F (HRTIM\_EEV8) |
| PA7 | Ethernet (ETH\_RX\_DV) |  | PB7 | HRTIM\_G (HRTIM\_EEV9) |
| PA8 |  |  | PB8 |  |
| PA9 |  |  | PB9 |  |
| PA10 |  |  | PB10 | SPI (SPI2\_SCK) |
| PA11 |  |  | PB11 |  |
| PA12 |  |  | PB12 | SPI (SPI2\_NSS) |
| PA13 | Debugging (SWDIO) |  | PB13 |  |
| PA14 | Debugging (SWCLK) |  | PB14 |  |
| PA15 | SPI (TRANSFER) |  | PB15 | SPI (SPI2\_MOSI) |
|  |  |  |  |  |
| **PORT C (8xDIO)** | |  | **PORT D (2xDIF, 12xDIO)** | |
| PC0 |  |  | PD0 |  |
| PC1 | Ethernet (ETH\_MDC) |  | PD1 |  |
| PC2 | Ethernet (ETH\_TXD2) |  | PD2 |  |
| PC3 | Ethernet (ETH\_TX\_CLK) |  | PD3 |  |
| PC4 | Ethernet (ETH\_RXD0) |  | PD4 |  |
| PC5 | Ethernet (ETH\_RXD1) |  | PD5 | HRTIM\_C (HRTIM\_EEV3) |
| PC6 |  |  | PD6 |  |
| PC7 |  |  | PD7 |  |
| PC8 |  |  | PD8 |  |
| PC9 |  |  | PD9 |  |
| PC10 | HRTIM\_A (HRTIM\_EEV1) |  | PD10 |  |
| PC11 |  |  | PD11 |  |
| PC12 | HRTIM\_B (HRTIM\_EEV2) |  | PD12 |  |
| PC13 |  |  | PD13 |  |
| PC14 |  |  | PD14 |  |
| PC15 |  |  | PD15 |  |
|  |  |  |  |  |
| **PORT E (2xDIFF, 12xDIO)** | |  | **PORT F (4x DIFF, 12xDIO)** | |
| PE0 |  |  | PF0 |  |
| PE1 |  |  | PF1 |  |
| PE2 | Ethernet (ETH\_TXD3) |  | PF2 |  |
| PE3 |  |  | PF3 |  |
| PE4 |  |  | PF4 |  |
| PE5 |  |  | PF5 |  |
| PE6 |  |  | PF6 |  |
| PE7 |  |  | PF7 |  |
| PE8 |  |  | PF8 |  |
| PE9 |  |  | PF9 |  |
| PE10 |  |  | PF10 |  |
| PE11 |  |  | PF11 |  |
| PE12 |  |  | PF12 |  |
| PE13 |  |  | PF13 |  |
| PE14 |  |  | PF14 |  |
| PE15 |  |  | PF15 |  |
|  |  |  |  |  |
| **PORT G (12xDIO)** | |  | **PORT H** | |
| PG0 |  |  | PH0 | Clock (RCC\_OSC\_IN) |
| PG1 |  |  | PH1 | Clock (RCC\_OSC\_OUT) |
| PG2 |  |  |
| PG3 |  |  |
| PG4 |  |  |
| PG5 |  |  |
| PG6 |  |  |
| PG7 |  |  |
| PG8 |  |  |
| PG9 |  |  |
| PG10 |  |  |
| PG11 | Ethernet (ETH\_TX\_EN) |  |
| PG12 | HRTIM\_D (HRTIM\_EEV5) |  |
| PG13 | Ethernet (ETH\_TXD0) |  |
| PG14 |  |  |
| PG15 |  |  |

Table 1. GPIO Map of the MCU.

*Note that this pinout may be subject to further modification during development.*

## Clock proposals

FTS375.