ARTIQ Sinara

2118 BNC-TTL / 2128 SMA-TTL

Features

- 8 TTL channels
- Input- and output-capable
- Galvanically isolated
- 3ns minimum pulse width
- BNC or SMA connectors

Applications

- Photon counting
- External equipment trigger
- Optical shutter control

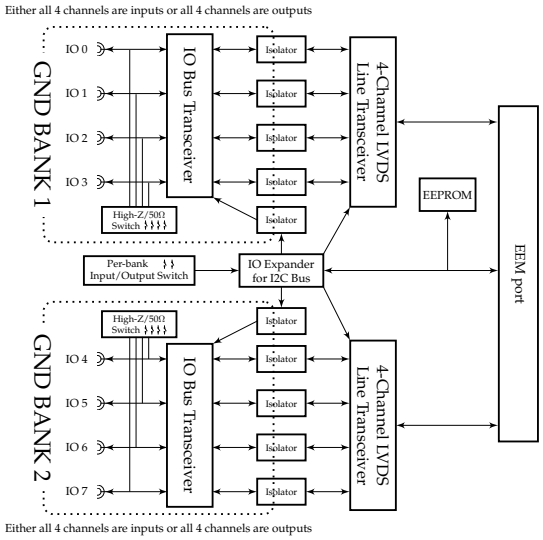


Figure 1: Simplified Block Diagram

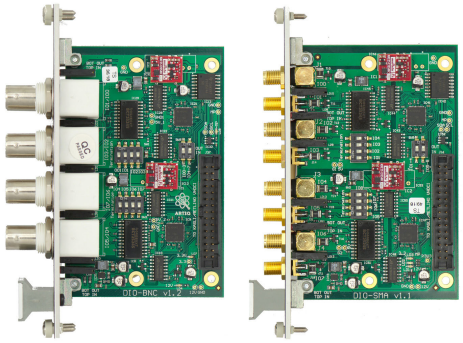


Figure 2: BNC-TTL and SMA-TTL cards

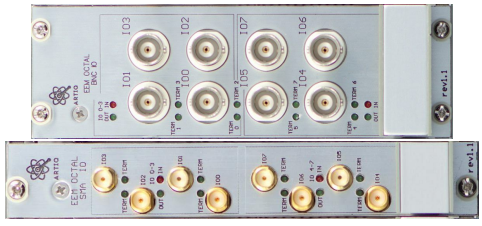


Figure 3: BNC-TTL and SMA-TTL front panels

General Description

The 2118 BNC-TTL card is an 8hp Sinara EEM; the 2128 SMA-TTL is a 4hp EEM. Both TTL cards are part of the ARTIQ/Sinara family and add general-purpose digital I/O capabilities to carrier cards such as 1124 Kasli and 1125 Kasli-SoC.

Each card provides two banks of four digital channels, for a total of eight digital channels, with respectively either BNC (2118) or SMA (2128) connectors. Each bank possesses individual ground isolation. The direction (input or output) of each bank can be selected using DIP switches, and applies to all four channels of the bank.

Each channel supports 50Ω terminations, individually controllable using DIP switches. Outputs tolerate short circuits indefinitely. Both cards are capable of a minimum pulse width of 3ns.

Isolated TTL cards are not well suited to low-noise or low-jitter applications due to interference from isolation components. For low-noise applications, use non-isolated cards such as 2238 MCX-TTL or 2245 LVDS-TTL.

Source

2118 BNC-TTL and 2128 SMA-TTL, like all the Sinara hardware family, are open-source hardware, and design files (schematics, PCB layouts, BOMs) can be found in detail at the repositories https://github.com/sinara-hw/DIO_BNC and https://github.com/sinara-hw/DIO_SMA.

Electrical Specifications

All specifications are in $0°C \leq T_A \leq 70°C$ unless otherwise noted. Specifications were derived based on the datasheets of the bus transceiver IC (SN74BCT25245DW[1]) and the isolator IC (SI8651BB-B-IS1[2]). The typical value of minimum pulse width is based on test results[3].

Table 1: Recommended Operating Conditions

Parameter	Min.	Typ.	Max.	Unit	Conditions
High-level input voltage[1]	2		5.5*	V	
Low-level input voltage[1]	-0.5		0.8	V	
Input clamp current[1]			-18	mA	termination disabled
High-level output current[1]			-160	mA	
Low-level output current[1]			376	mA	

*With the 50Ω termination enabled, the input voltage should not exceed 5V.

Table 2: Electrical Characteristics

Parameter	Min.	Typ.	Max.	Unit	Conditions
High-level output voltage[1]	2			V	I_{OH}=-160mA
	2.7			V	I_{OH}=-6mA
Low-level output voltage[1]		0.42	0.55	V	I_{OL}=188mA
			0.7	V	I_{OL}=376mA
Minimum pulse width[2,3]		3	5	ns	
Pulse width distortion[2]		0.2	4.5	ns	
Peak jitter[2]		350		ps	
Data rate[2]	0		150	Mbps	

Low-jitter applications should note carefully the jitter introduced by the signal isolator. Noise is also introduced between the primary and secondary domains by the DC/DC converter. Where noise or jitter are crucial, it is instead recommended to use non-isolated cards such as 2238 MCX-TTL or 2245 LVDS-TTL.

Minimum pulse width was measured by generating pulses of progressively longer duration through a DDS generator and using them as input for a BNC-TTL card. The input BNC-TTL card was connected to another BNC-TTL card as output. The output signal is measured and shown in Figure 4.

[1] https://www.ti.com/lit/ds/symlink/sn74bct25245.pdf
[2] https://www.skyworksinc.com/-/media/Skyworks/SL/documents/public/data-sheets/si865x-datasheet.pdf
[3] https://github.com/sinara-hw/sinara/issues/187

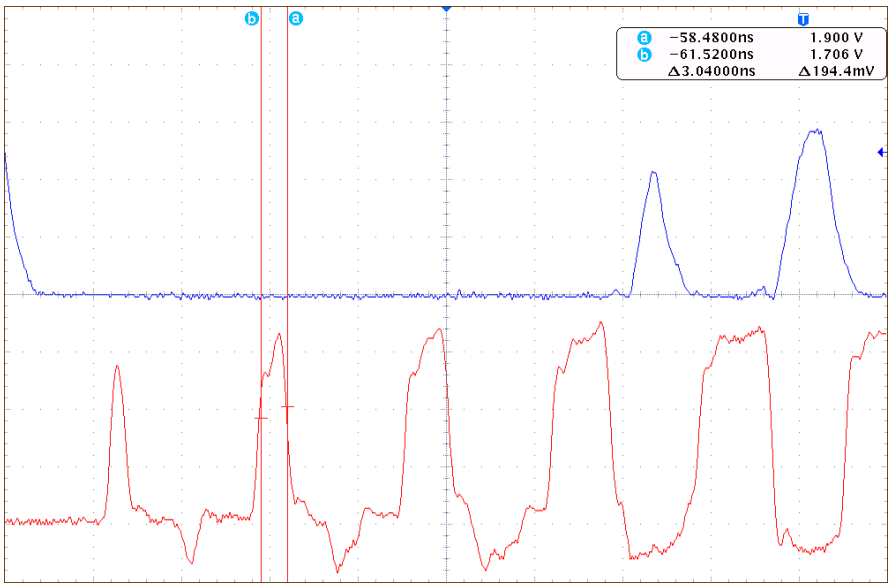


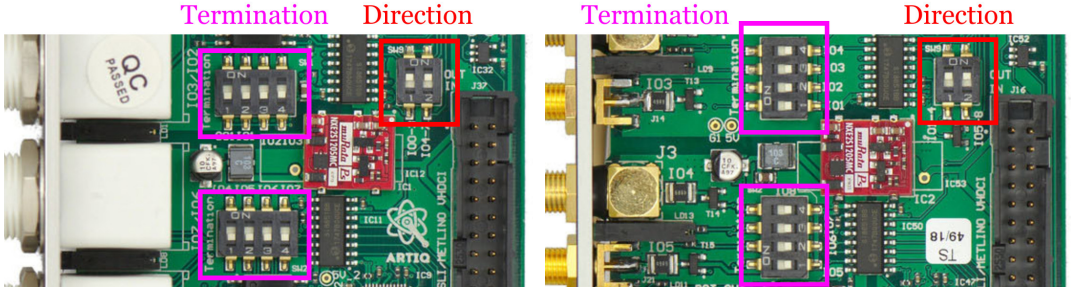
Figure 4: Minimum pulse width required for BNC-TTL card

The red trace shows the DDS generator input pulses. The blue trace shows the measured signal from the output BNC-TTL. Note that the first red pulse failed to reach the 2.1V threshold required by TTL and was not propagated. The first blue (output) pulse is the result of the second red (input) pulse, of 3ns width, which propagated correctly.

Configuring IO Direction & Termination

IO direction and termination must be configured by setting physical switches on the board. The termination switches are found on the middle-left and the IO direction switches on the middle-right of both cards. Termination switches select between high impedance (`OFF`) and 50Ω (`ON`). Note that termination switches are by-channel but IO direction switches are by-bank.

- IO direction switch closed (`ON`)
  Fixes the corresponding bank to output. The IO direction cannot be changed by I^2C.
- IO direction switch open (`OFF`)
  The corresponding bank is set to input by default. IO direction *can* be changed by I^2C.



(a) BNC-TTL (b) SMA-TTL

Figure 5: Position of switches

ARTIQ System Description Entry

ARTIQ/Sinara firmware/gateware is generated according to a JSON system description file, allowing gateware to be specific to and optimized for a certain system configuration.

2118 BNC-TTL and 2128 SMA-TTL should be entered in the `peripherals` list of the corresponding core device in the following format:

```
"name" : {
    "type": "dio",
    "board": "DIO_BNC", // or "DIO_SMA", optional
    "ports": [0],
    "edge_counter": true, // optional
    "bank_direction_low": "input", // or "output"
    "bank_direction_high": "output" // or "input"
}
```

Replace 0 with the EEM port number used on the core device. Any port can be used. The `edge_counter` field is boolean and may be specified true or false; a setting `true` will make a corresponding ARTIQ `edge_counter` module available and consume a corresponding amount of additonal gateware resources. If not included, its default value is false.

Example ARTIQ Code

The sections below demonstrate simple usage scenarios of extensions on the ARTIQ control system. These extensions make use of the resources of the 2118 BNC-TTL/2128 SMA-TTL cards. They do not exhaustively demonstrate all the features of the ARTIQ system.

The full documentation for ARTIQ software and gateware, including guides for their use, is available at https://m-labs.hk/artiq/manual/. Please consult the manual for details and reference material of the functions and structures used here.

Timing accuracy in these examples is well under 1 nanosecond thanks to ARTIQ RTIO infrastructure.

One pulse per second

The channel should be configured as output in both the gateware and hardware.

```
@kernel
def run(self):
    self.core.reset()
    while True:
        self.ttl0.pulse(500*ms)
        delay(500*ms)
```

Morse code

This example demonstrates some basic algorithmic features of the ARTIQ-Python language.

```
def prepare(self):
    # As of ARTIQ-6, the ARTIQ compiler has limited string handling
    # capabilities, so we pass a list of integers instead.
    message = ".- .-. - .. --.-"
    self.commands = [{".": 1, "-": 2, " ": 3}[c] for c in message]

@kernel
def run(self):
    self.core.reset()
    for cmd in self.commands:
        if cmd == 1:
            self.led.pulse(100*ms)
            delay(100*ms)
        if cmd == 2:
            self.led.pulse(300*ms)
            delay(100*ms)
        if cmd == 3:
            delay(700*ms)
```

Sub-coarse-RTIO-cycle pulse

With the use of ARTIQ RTIO, only one event can be enqueued per *coarse RTIO cycle*, which typically corresponds to 8ns. To emit pulses of less than 8ns, careful timing is needed to ensure that the `ttl.on()` & `ttl.off()` event are submitted during different coarse RTIO cycles.

```
@kernel
def run(self):
    self.core.reset()
    delay(6*ns)             # Coarse RTIO period: 0 - 7  ns
    self.ttl0.pulse(3*ns)   # Coarse RTIO period: 8 - 15 ns
```

Edge counting in a 1ms window

The `TTLInOut` class implements `gate_rising()`, `gate_falling()` & `gate_both()` for rising edge, falling edge, both rising edge & falling edge detection respectively. The channel should be configured as input in both gateware and hardware. Invoke one of the 3 methods to start edge detection.

```
    # Start input gate & advance timeline cursor to gate_end_mu
    gate_end_mu = self.ttlin.gate_rising(1*ms)
```

Input signal can generated from another TTL channel or from other sources. Manipulate the timeline cursor to generate TTL pulses using the same kernel.

```
@kernel
def run(self):
    self.core.reset()
    gate_start_mu = now_mu()
    # Start input gate & advance timeline cursor to gate_end_mu
    gate_end_mu = self.ttlin.gate_rising(1*ms)
    at_mu(gate_start_mu)

    for _ in range(64):
        self.ttlout.pulse(8*ns)
        delay(8*ns)

    counts = self.ttlin.count(gate_end_mu)
```

The detected edges are registered to the RTIO input FIFO. By default, the FIFO can hold 64 events. The FIFO depth is defined by the `ififo_depth` parameter for `Channel` class in `rtio/channel.py`. Once the threshold is exceeded, an `RTIOOverflow` exception will be triggered when the input events are read by the kernel CPU. Finally, invoke `count()` to retrieve the edge count from the input gate.

The RTIO system can report at most one edge detection event for every coarse RTIO cycle. In principle, to guarantee all rising edges are counted (with `gate_rising()` invoked), the theoretical minimum separation between rising edges is one coarse RTIO cycle (typically 8 ns). However, both the electrical specifications and the possibility of triggering `RTIOOverflow` exceptions should also be considered.

Edge counting using `EdgeCounter`

This example code uses a gateware counter to substitute the software counter, which has a maximum count rate of approximately 1 million events per second. If a gateware counter is enabled on the TTL channel, it can typically count up to 125 million events per second:

```
@kernel
def run(self):
    self.core.reset()
    self.edgecounter0.gate_rising(1*ms)
    counts = self.edgecounter0.fetch_count()
    print(counts)
```

Edges are detected by comparing the current input state and that of the previous coarse RTIO cycle. Therefore, the theoretical minimum separation between 2 opposite edges is 1 coarse RTIO cycle (typically 8 ns).

Responding to an external trigger

One channel needs to be configured as input, and the other as output.

```
@kernel
def run(self):
    self.core.reset()
    gate_end_mu = self.ttlin.gate_rising(5*ms)
    timestamp_mu = self.ttlin.timestamp_mu(gate_end_mu)
    at_mu(timestamp_mu + self.core.seconds_to_mu(10*ms))
    self.ttlout.pulse(1*us)
```

62.5 MHz clock signal generation

A TTL channel can be configured as a `ClockGen` channel, which generates a periodic clock signal. Each channel has a phase accumulator operating on the RTIO clock, where it is incremented by the frequency tuning word at each coarse RTIO cycle. Therefore, jitter should be expected when the desired frequency cannot be obtained by dividing the coarse RTIO clock frequency with a power of 2.

Typically, with the coarse RTIO clock at 125 MHz, a `ClockGen` channel can generate up to 62.5 MHz.

```
@kernel
def run(self):
    self.core.reset()
    self.ttl0.set(62.5*MHz)
```

Minimum sustained event separation

The minimum sustained event separation is the least time separation between input gated events for which all gated edges can be continuously & reliably timestamped by the RTIO system without causing `RTIOOverflow` exceptions. The following `run()` function finds the separation by approximating the time of running `timestamp_mu()` as a constant. Import the `time` library to use `time.sleep()`.

```
@kernel
def get_timestamp_duration(self, pulse_num) -> TInt64:
    self.core.break_realtime()
    delay(1*ms)
    gate_start_mu = now_mu()
    # Start input gate & advance timeline cursor to gate_end_mu
    gate_end_mu = self.ttlin.gate_rising(1*ms)
    at_mu(gate_start_mu)

    self.ttlclk.set_mu(0x800000)
    delay(16*pulse_num*ns)
    self.ttlclk.set_mu(0)

    # Guarantee t0 > gate_end_mu
    # Otherwise timestamp_mu may wait for pulses till gate_end_mu
    rtio_time_mu = self.core.get_rtio_counter_mu()
    sleep_mu = float(gate_end_mu - rtio_time_mu)
    self.rpc_sleep(self.core.mu_to_seconds(sleep_mu))

    t0 = self.core.get_rtio_counter_mu()
    while self.ttlin.timestamp_mu(gate_end_mu) >= 0:
        pass
    t1 = self.core.get_rtio_counter_mu()
    return t1 - t0

@rpc
def rpc_sleep(self, duration):
    time.sleep(duration)

@kernel
def run(self):
    self.core.reset()
    t64 = self.get_timestamp_duration(64)
    t8 = self.get_timestamp_duration(8)
    print("Mean timestamp_mu duration:")
    print(self.core.mu_to_seconds((t64 - t8)/((64 + 1) - (8 + 1))))
```

Table 3: Minimum sustained event separation of different carriers

Carrier	Kasli v1.1	Kasli-SoC
Duration	650 ns	600 ns

Ordering Information

To order, please visit https://m-labs.hk and choose 2118 BNC-TTL/2128 SMA-TTL in the ARTIQ/Sinara hardware selection tool. Cards can be ordered as part of a fully-featured ARTIQ/Sinara crate or standalone through the 'Spare cards' option. Otherwise, orders can also be made by writing directly to mailto:sales@m-labs.hk.