Advanced **L**ogic **S**ynthesis for **E**lectronics
http://www.alse-fr.com

v2025.04g

ALSE's GEDEK
(Gigabit Ethernet Data Exchange Kit)

I2C & Ethernet Reference Design for Lattice MachXO5

Preamble

A very long time ago, we (A.L.S.E.) have invented the concept of a processor-less Ethernet stack !

We named the IP "**GEDEK**" for **Gigabit Ethernet Data Exchange Kit** and it has since helped a lot of customers to implement high speed data transfers in their FPGA designs using minimal resources.

If you're not already familiar with the concept, please read the Introduction to GEDEK document available on our web site.

We have ported this IP to practically all FPGAs that exist, bringing ultimate performance and tiny footprint to all FPGA users. We also support all kinds of Ethernet interfaces and PHYs, including SERDESes (transceivers) and LVDS (with CDR).
GEDEK also works on 100M ("Fast") Ethernet, and we have recently added a **2.5G Ethernet** option.
Logically, we have extended the concept to **10G Ethernet** with our "**10GEDEK**" IP.

Since its early days, we have added a lot of new features and options to the original GEDEK which has been copied but remains unrivaled in terms of performance and features.

GEDEK (including our proprietary MACs) is **very compact** and can fit in a portion of the smallest FPGAs, and indeed in the Lattice MachXO5-25 FPGA, with a lot of room left for application logic.

This Reference Design shows an application where a small FPGA (MachXO5-25) can host 14 independent I2C controllers running in parallel, while ensuring bi-directional data transfer with a PC through Ethernet, within fraction of the FPGA. The ALSE I2C controller IP is very compact, which allows to use many instances in parallel.

In this example, the PC receives I2C data automatically and it controls the FPGA by reading and writing in the FPGA space (registers map and memories) through Ethernet.

I2C & Ethernet Reference Design Principle

A **Lattice MachXO5T-NX development board** is used with up to 4 x **Pmod I2C Temperature sensors** attached. Each PMOD connector can host two sensor modules.

The design configures the sensors in their most accurate (and slowest) mode (12 bits with a resolution of 1/16th Celsius). The sensors are read **4 times per second**.

The LEDs can display the LSBits of Sensor 0 temperature, or a value written by the PC.

The LEDs are dimmed by a PWM which can be adjusted by the PC.

We provide a Python utility program : **i2c_eth.py** to exercise the FPGA.

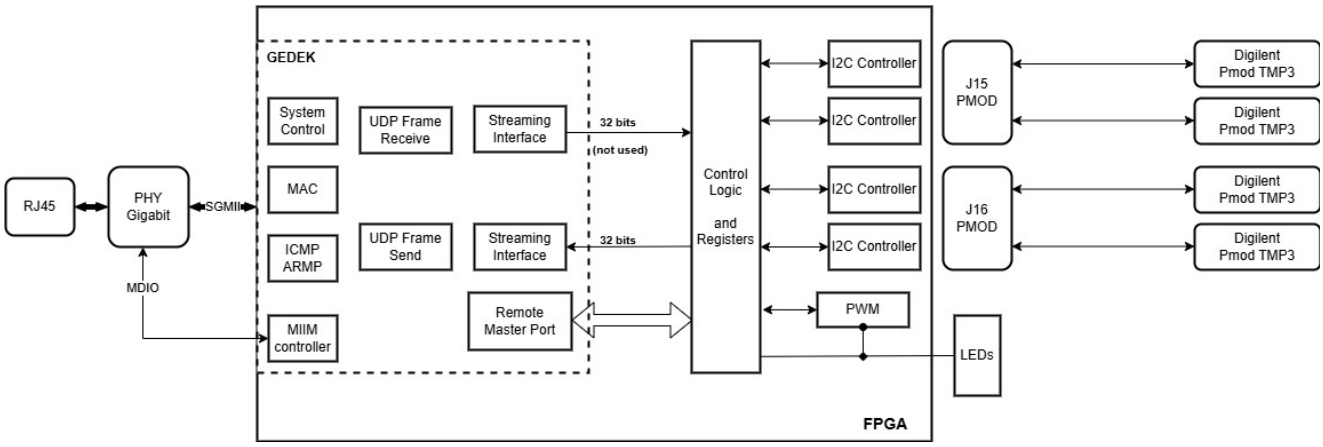
We also provide a command line utility **gedek_rw.exe** to read and write in the FPGA register map.

What you need

To implement and test the *I2C GEDEK Reference Design*, you need :

- A Lattice MachXO5T-NX Development Board with the mini-USB cable
- One to four Digilent Pmod TMP3 I2C temperature sensors (easy to find with Google)
- **Male-to-Female** 10 cm Dupont cables (4 wires per Temperature module)
- An RJ45 cable (cat 5e or better)
- a PC under Windows or Linux with a wired Gigabit Ethernet interface.
- Lattice Radiant software installed (or at least the Radiant Standalone Programmer).
- Optional : a USB3 ↔ Gigabit Ethernet adapter

Block Diagram of the Reference Design



This is a simplified view of the FPGA and GEDEK contents.

The RJ45 connector is attached (through a transformer not represented here) to a physical interface chip aka “PHY”. In this case the PHY is a GPY115C0VI from Maxlinear.
Note that **GEDEK works with all PHYs !**

The PHY chip is connected to the FPGA through industry-standard interfaces :

- As **Data** link, this PHY uses **SGMII** (high-speed serial protocol).
  In this case, it is wired to **LVDS** inputs controlled by our own PCS, *without using a transceiver* nor the Lattice IP.
- For **Control** (access to the PHY's control and status registers), we implement and support the standard **MDIO** interface. In most cases, we don’t even need to use this link, but due to the design of board, we had to use our MIIM controller to control the PHY and make sure it does automatically connect to the network.

Important ! Due to a design / manufacturing error, some boards are not correctly configured and will not succeed to connect to the network. If you experience issues see the “Tap Up” paragraph to solve the problem.

Ethernet Network

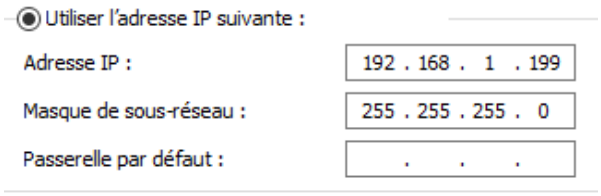
With the Reference Design, you will be able to control the FPGA and exchange data from and to PC(s) located anywhere on your local network. It will work perfectly and automatically even if you cross several Ethernet routers and switches.
Why ? Because GEDEK is not just a local point-to-point solution ! It is capable of establishing and maintaining logical connections between the FPGA and an external target.

For the default reference design, **make sure your PC is on the following network: 192.168.1.xx** and verify that **192.168.1.18** is available (this is the GEDEK IP address).
If 192.168.1.18 is not available, displace the offender ;-)
If you are in a different network, configure your PC in Fixed IP mode like 192.168.1.**99** (for example), it can be at another address than .99 but make sure you **avoid .18** indeed !

Useful Hint

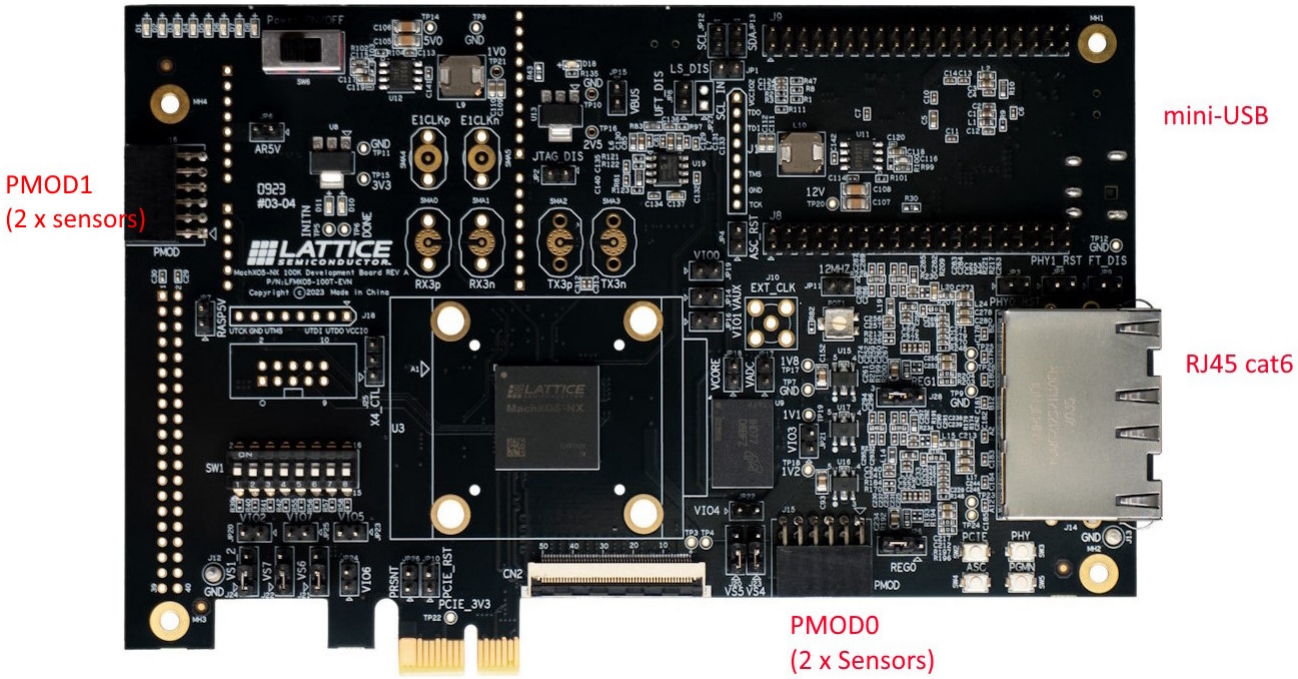
If your PC has only one Gigabit adapter used for your main connection, and especially if your current network is not 192.168.1.xx, it may be best to use a cheap USB3 Ethernet adapter that will provide an independent Gigabit Ethernet network that you will manually assign to 192.168.1.xx.

- Plug the adapter to a USB3 port.
- Open the properties of this adapter in your Network Connections panel.
- "Internet Version 4 (TCP/IPv4)" – Properties :



- Apply.
- Connect the Ethernet adapter to the MachXO5 port (on the middle RJ45) : it should synchronize in no more than one second. If not, see next page.
- On your PC : disconnect your main network Ethernet cable.
- `ping 192.168.1.18` : the MachXO5 should answer
- You can now run the provided utility programs.

Preparing the Lattice MachXO5T-NX Development Board



Note ! You do not have to use the 12V power adapter : the USB port used to program the FPGA is sufficient to power the board reliably.

- Connect a mini-USB cable to the **Config USB Header J11** (bottom side)
- Connect an **RJ45** (cat6) cable to the indicated RJ45 connector (closest to the USB)
- Connect the other side of the cable to a Gigabit Ethernet switch
- Turn on the board with the **PW6** "Power ON/OFF) power switch (right position).
- **Program** the on-board Flash with the FPGA design as detailed below.
- Perform a Power Cycle.
- Verify that the indicator *on your Ethernet switch** turns on, indicating a successful synchronization at 1Gb/s. **If it does not, read next page !**
  * No LED will show on the FPGA RJ45.

Board design issues

The MachXO5T-NX Development board suffers from several design issues that are not definitive show-stoppers but some need to be mitigated before using some features.
If you plan to copy part of this board for your own product, please ask for our report which lists the problems we have identified !

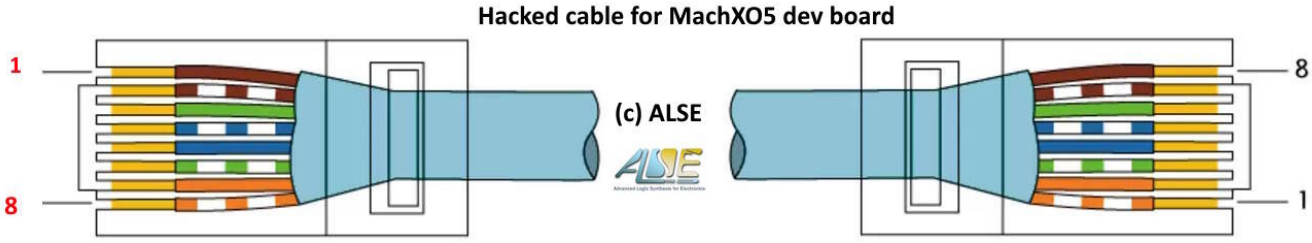
If your board is hit by the “Tap Up” problem (0R soldered on R170 and R213), here is one of the two solutions that we recommend.

Working around the “Tap Up” problem

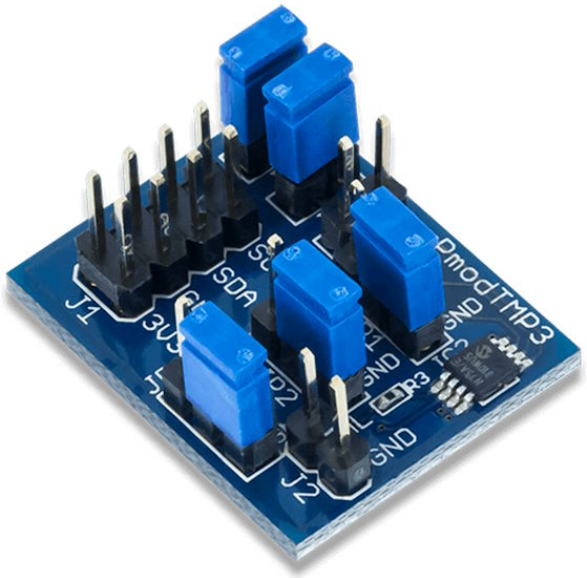
Solution 2 : create one or two **custom RJ 45 cable(s)** like below. One side (the right one below) is wired as usual, the second (the left one below) is attached “backwards”.

Make sure you place a **big visible tag** on these cables ! ***Don’t use them for anything else***.

Note that this cable is **not** a “cross-over” cable.



Wiring the PMOD I2C Temperature sensors



As you can see, the pins are assigned so the connection is straight between the PMOD connector and the sensors pins (J1 left or right row). Use four Male-to-Female Dupond wires.
Be careful to respect the cabling ! You could damage the sensor or the FPGA board.

The jumpers must keep their default positions shown above.

You can connect up to four sensors (two per PMOD).

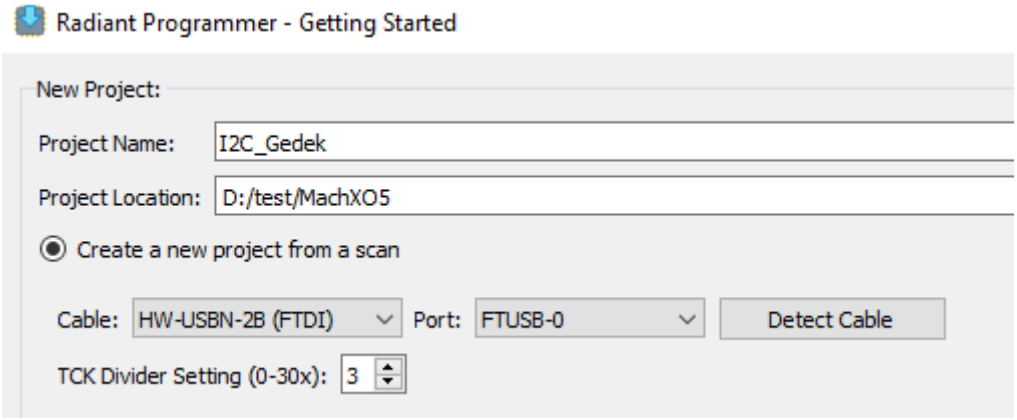
Programming the on-board Flash

You can program the FPGA using the provided bitstream (.bit), but you'll have to do this every time the board is powered up.

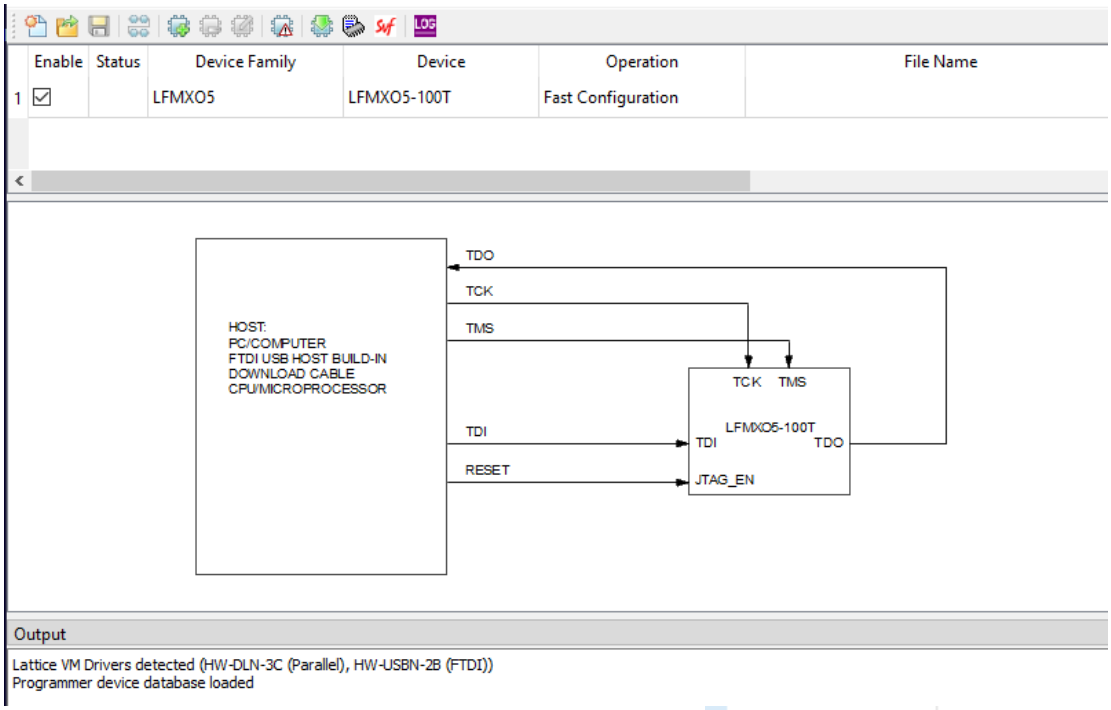
Alternatively, you can program the on-board Flash once, and the FPGA will automatically load the I2C Reference Design when it is powered up.

Here is how to do this :

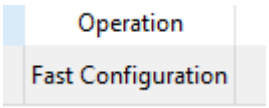
- Prepare the three programming files provided. We will use the **.mcs** and the **.jed** files.
  As mentioned above, the .bit file is for SRam programming.
- Launch the **Radiant Programmer**.
- Create a new project like below.
  You can "Detect" the programming cable, be sure to choose the correct port (**FTUSB-0**).



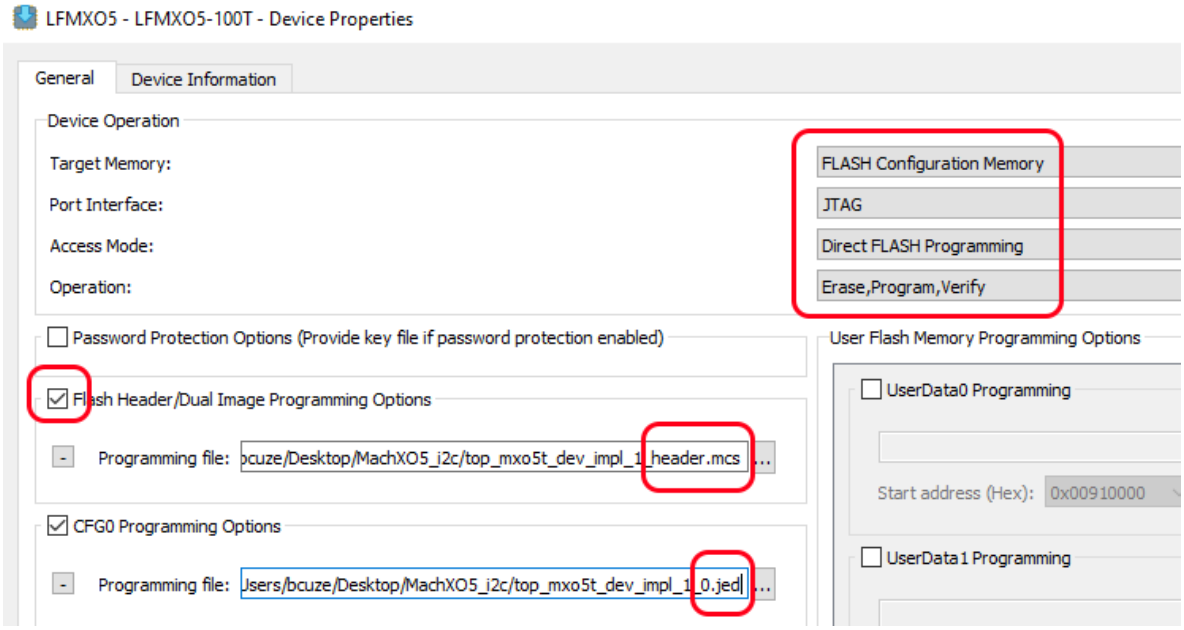
- After a few seconds you should see that the FPGA has been recognized.



- Double-click on "Fast Configuration"



- Use the settings below to program the on-board Flash
- Use the .mcs and the .jed files that you have prepared as shown above. Then select "Ok".



- Once you are ready, click on the “Program Device” button
  It should take less than one minute to complete.
- Turn the power off, then back on :
  the LEDs should display the seven LSBs of the temperature acquired on Sensor 0.
  The rightmost LED does toggle every second.

Interacting with the board

Python script

`python i2c_eth.py -h` does display the commands available :

```
usage: i2c_eth.py [-h] [-i IP_ADDRESS] [-f RECORD_FILE] [-l LED] [-p PWM]
[-t] [-rp REG_PORT] [-dp DATA_PORT]

PC tool for I2C over Ethernet.

options:
  -h, --help            show this help message and exit
  -i IP_ADDRESS, --ip-address IP_ADDRESS
                        IP address of the board (default: 192.168.1.18)
  -f RECORD_FILE, --record-file RECORD_FILE
                        Record file name prefix (default: adc_data)
  -l LED, --led LED     Write the LED register (default: None)
  -p PWM, --pwm PWM     Write the PWM register (default: None)
  -t, --temp-led        Restore LEDs to temperature (default: False)
  -rp REG_PORT, --reg-port REG_PORT
                        Register interface UDP port (default: 1235)
  -dp DATA_PORT, --data-port DATA_PORT
                        Data interface UDP port (default: 1236)
```

Testing the board

- Make sure your sensor(s) is/are connected as explained.
- Make sure Python3 is installed on your PC
- Open a command shell in the "**`/soft`**" directory
- **`python i2c_eth.py -p 255`**
  The LEDs should become brighter (PWM to max)
  and the temperatures (4 sensors) should be displayed continuously :

```
--- ALSE Ethernet I2C Tool ---
    (c) ALSE - info@alse-fr.com - Version 2025.04.18
   Using IP address: 192.168.1.18, UDP register & data ports: #1235/#1236

FPGA Version = 2025.04.18
Writing 0xFF to the PWM register...
Receiving temperature data (Ctrl+C to interrupt)...
T0 = -15.6 °C / T1 = 24.2 °C / T2 = ---- °C / T3 = ---- °C
```

Notes !
* A value of **`----°C`** indicates no sensor. In our case, two sensors were connected to PMOD0.
* We display negative values as they should (we used an anti-dust spray to cool sensor 0).

- Dim the LEDs to the minimum brightness :

`python i2c_eth.py -p 1`

- Adjust to brighter :

`python i2c_eth.py -p 200`

Note : this utility handles **decimal values** (like 255 eg).
We'll see next that the CLI tool handles hexadecimal values.

Read/Write registers CLI utility

We have a command line (executable) utility to read and write FPGA registers, both for Windows (example below) and for Linux.

Open a command shell and run **gedek_rw.exe**.

User Commands are :

- '**R**' : Read an FPGA Register.
  e.g : `r 0` => read the FPGA register #0 (Version).
- '**W**' : Write an FPGA Register (register address and data values are **in *hexadecimal*)**
  e.g : `w 3 8F` => write the register located @ address 0x3 with the value 0x8F (PWM~=56%)
- '**Q**' : Quit the application

Pinging is important !

The software utilities all start by PINGing the FPGA.
This is important because the FPGA (GEDEK) uses the PING *sender* information as a destination for the frames sent by the FPGA. This “automatic” mode is very handy (but it can be disabled).

If you PING the FPGA from another PC than the one running the utility, the frames sent by the FPGA will be routed to the new PC !

FPGA Registers

Available Registers, that can be accessed using the (single) Read/Write functions 'w' and 'r', are :

Address	Mode	Reset Value	Description
0x0	RO	0x2025xxxx	FPGA Version, xxxx is Month and Day of Ref Design version. Eg : 20250307 for a Ref Design version from 7th of March 2025
0x1	RW	0x00000001	Read : Status (not used) Write : LEDs controlled by sensor 0
0x2	RW	0x00000000	RW Register that you can write and read back, to check Write & Read accesses. Once written into, this register drives the LEDs
0x3	RW	0x00000080	PMW (0 ..255) RW Register that you can write and read back The PWM ratio dims the LEDs
0x4	RO	-	Sensor 0 Temperature
0x5	RO	-	Sensor 1Temperature
0x6	RO	-	Sensor 2 Temperature
0x6	RO	-	Sensor 3 Temperature
Others	RO	n.a.	0xdeadbeef

RO = Read-Only, RW = Read/Write

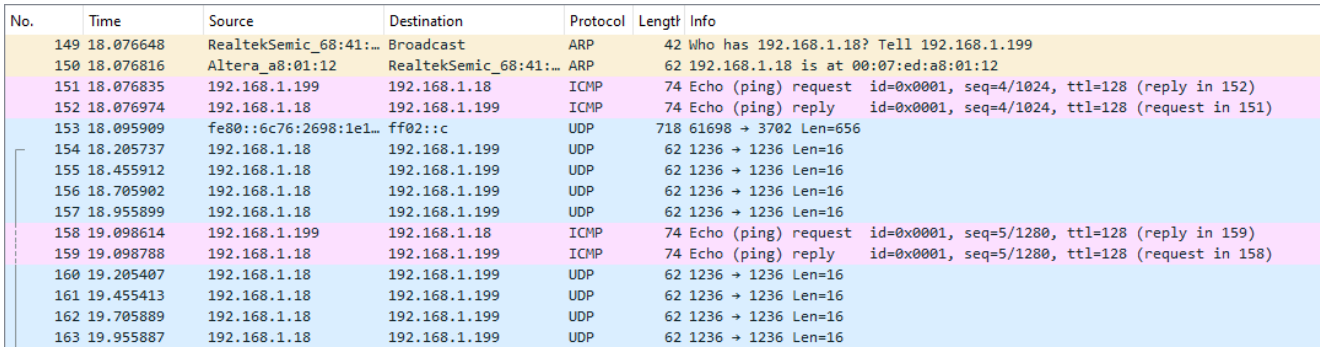
The **Status Register** mapping is :

Bits	Reset Value	Description
0	1	PLL Lock signal (not very useful)
7:1	0	Unused
31:8	0	Reserved

Debugging using a Protocol Analyzer : Wireshark

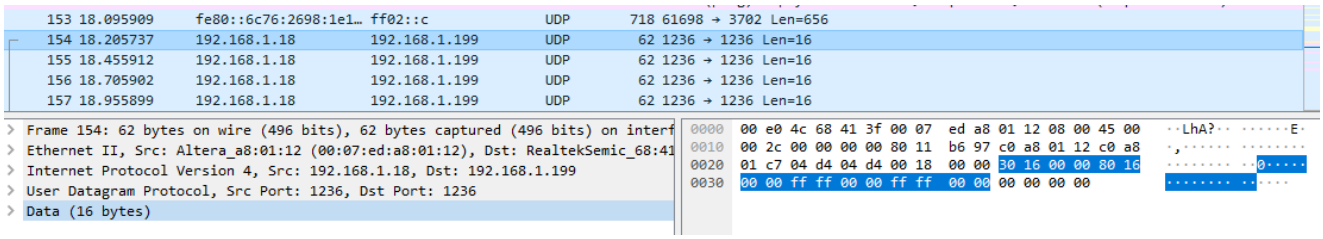
You may fear that debugging your future Ethernet application is going to be extremely difficult due to the complexity of the Ethernet protocol, the physical coding, and the voltage levels. Don't ! Your PC can easily be turned into a powerful Ethernet Protocol Analyzer, at zero cost, using the excellent open source tool named **WireShark**, which you can find at http://www.wireshark.org. All the documentation is available at this site as well, so we won't try to replace it here. We just provide a very quick and superficial how-to, in order to quickly visualize the frames we exchange with our Reference Design. Before stepping through these simplified instructions, we recommend you look at a video introduction.

- Download **Wireshark** from the above Web site and install it.
  During the installation, confirm that you want to install WinPcap (this is required).
  You may have to restart your computer.
- Make sure the FPGA board is programmed (with the Reference Design) and connected.
- Launch **Wireshark**
- Select the Network adapter connected to the MachXO5 board.
- Start the Wireshark capture.
- Open a command shell and :
  ping 192.168.1.18
  this must succeed. Otherwise, test the connection and the network settings.
- After a few seconds, stop the capture.
  With Wireshark, you see the actual Internet Frames, complete with header etc. The only thing you don't see are the preambles and postambles for the physical frames, which are stripped down by the PHY (both in your PC and in the FPGA board).
  Wireshark does also interpret (“dissect”) the frame and show its contents.



No.	Time	Source	Destination	Protocol	Length	Info
149	18.076648	RealtekSemic_68:41:…	Broadcast	ARP	42	Who has 192.168.1.18? Tell 192.168.1.199
150	18.076816	Altera_a8:01:12	RealtekSemic_68:41:…	ARP	62	192.168.1.18 is at 00:07:ed:a8:01:12
151	18.076835	192.168.1.199	192.168.1.18	ICMP	74	Echo (ping) request id=0x0001, seq=4/1024, ttl=128 (reply in 152)
152	18.076974	192.168.1.18	192.168.1.199	ICMP	74	Echo (ping) reply id=0x0001, seq=4/1024, ttl=128 (request in 151)
153	18.095909	fe80::6c76:2698:1e1…	ff02::c	UDP	718	61698 → 3702 Len=656
154	18.205737	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
155	18.455912	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
156	18.705902	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
157	18.955899	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
158	19.098614	192.168.1.199	192.168.1.18	ICMP	74	Echo (ping) request id=0x0001, seq=5/1280, ttl=128 (reply in 159)
159	19.098788	192.168.1.18	192.168.1.199	ICMP	74	Echo (ping) reply id=0x0001, seq=5/1280, ttl=128 (request in 158)
160	19.205407	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
161	19.455413	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
162	19.705889	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
163	19.955887	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16

We see the ping requests answered by GEDEK as well as the Temperature frames generated by the reference design :



153	18.095909	fe80::6c76:2698:1e1…	ff02::c	UDP	718	61698 → 3702 Len=656
154	18.205737	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
155	18.455912	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
156	18.705902	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16
157	18.955899	192.168.1.18	192.168.1.199	UDP	62	1236 → 1236 Len=16

```
> Frame 154: 62 bytes on wire (496 bits), 62 bytes captured (496 bits) on interf
> Ethernet II, Src: Altera_a8:01:12 (00:07:ed:a8:01:12), Dst: RealtekSemic_68:41
> Internet Protocol Version 4, Src: 192.168.1.18, Dst: 192.168.1.199
> User Datagram Protocol, Src Port: 1236, Dst Port: 1236
> Data (16 bytes)
```

```
0000  00 e0 4c 68 41 3f 00 07  ed a8 01 12 08 00 45 00   ··LhA?·· ·····E·
0010  00 2c 00 00 00 00 80 11  b6 97 c0 a8 01 12 c0 a8   ·,····· ········
0020  01 c7 04 d4 04 d4 00 18  00 00 30 16 00 00 80 16   ········ ··0·····
0030  00 00 ff ff 00 00 ff ff  00 00 00 00 00 00        ········ ······
```

The 32 bits words are for sensor 0 : “30 16 00 00”
which actually is 0x00001630,
which is **22.1875°C** (16h = 22d, 30h=3/16=0.1875)

You can also capture the frames exchanged when using the Read and Write commands.

This concludes this very simple introduction to Wireshark.

GEDEK I2C Reference Design Compilation results

We have compiled a configuration that includes **14 (fourteen) x I2C controllers**.
To make sure nothing was optimized away, we have recombined all the 14 SDA and 14 SCL to output on a Spare pin. So this estimation is reliable.

Here is the map report summary :

Map

```
                        Lattice Mapping Report File

Design:  TOP
Family:  LFMXO5
Device:  LFMXO5-100T
Package: BBG400
Performance Grade:  9_High-Performance_1.0V

Mapper:    version Radiant Software (64-bit) 2024.1.1.259.1
Mapped on: Tue Apr 22 09:30:48 2025
```

Design Summary

```
   Number of registers:         3826 out of 80745 (5%)
      Number of SLICE         registers: 3825 out of 79872 (5%)
      Number of PIO Input     registers:    0 out of   291 (0%)
      Number of PIO Output    registers:    0 out of   291 (0%)
      Number of PIO Tri-State registers:    1 out of   291 (<1%)
   Number of LUT4s:            8837 out of 79872 (11%)
      Number used as logic LUT4s:                      6183
      Number used as distributed RAM:                  1944 (6 per 16X4 RAM)
      Number used as ripple logic:                      710 (2 per CCU2)
```

and the detailed report :

Map Resource Usage

```
#Map Resource Utilization Report file generated by Lattice Radiant Software (64-bit) 2024.1.1.2
#Generated on 04/22/25 09:30:56
#DESIGN = top_mxo5t_dev
#DEVICE = LFMXO5-100T
#PACKAGE = BBG400
#OPERATING = Industrial
#PERFORMANCEGRADE = 9_High-Performance_1.0V
```

	LUT4 Logic	LUT4 Distributed RAM	LUT4 Ripple Logic	PFU Registers	IO Registers	IO Buffers	EBR
▼ TOP	6183(604)	1944(0)	710(160)	3825(1012)	1(1)	44(44)	4(0)
▶ g_gedek.gedek_sgmii_if_lattice_inst	273(0)	0(0)	0(0)	60(0)	0(0)	0(0)	0(0)
▶ g_gedek.i_gedek	2986(126)	1944(0)	370(56)	1675(626)	0(0)	0(0)	4(0)
▼ g_gedek.miim_top_inst	61(15)	0(0)	40(24)	66(26)	0(0)	0(0)	0(0)
iMIIM	46(46)	0(0)	16(16)	40(40)	0(0)	0(0)	0(0)
▼ gg.0.i2c	130(63)	0(0)	8(0)	70(25)	0(0)	0(0)	0(0)
bit_ctrl	67(67)	0(0)	8(8)	45(45)	0(0)	0(0)	0(0)
▼ gg.1.i2c	130(63)	0(0)	8(0)	70(25)	0(0)	0(0)	0(0)
bit_ctrl	67(67)	0(0)	8(8)	45(45)	0(0)	0(0)	0(0)
▼ gg.10.i2c	135(68)	0(0)	8(0)	70(25)	0(0)	0(0)	0(0)
bit_ctrl	67(67)	0(0)	8(8)	45(45)	0(0)	0(0)	0(0)
▼ gg.11.i2c	130(63)	0(0)	8(0)	70(25)	0(0)	0(0)	0(0)
bit_ctrl	67(67)	0(0)	8(8)	45(45)	0(0)	0(0)	0(0)

GEDEK with its options represents about 5000 LUTs
Each I2C controller approximately costs between 162 and 277 LUTs.

The complete system with 14 x I2C controllers and Ethernet occupies about 10% of the MachXO5-100.

Conclusion

This Reference Design should demonstrate the simplicity and the performance of the GEDEK solution especially when associated with optimized implementation of other functions like I2C.

If you have any question, please contact us !

Contact : info@alse-fr.com

The A.L.S.E. Team

***A**dvanced **L**ogic **S**ynthesis for **E**lectronics*