FPGA Firmware Progress Log

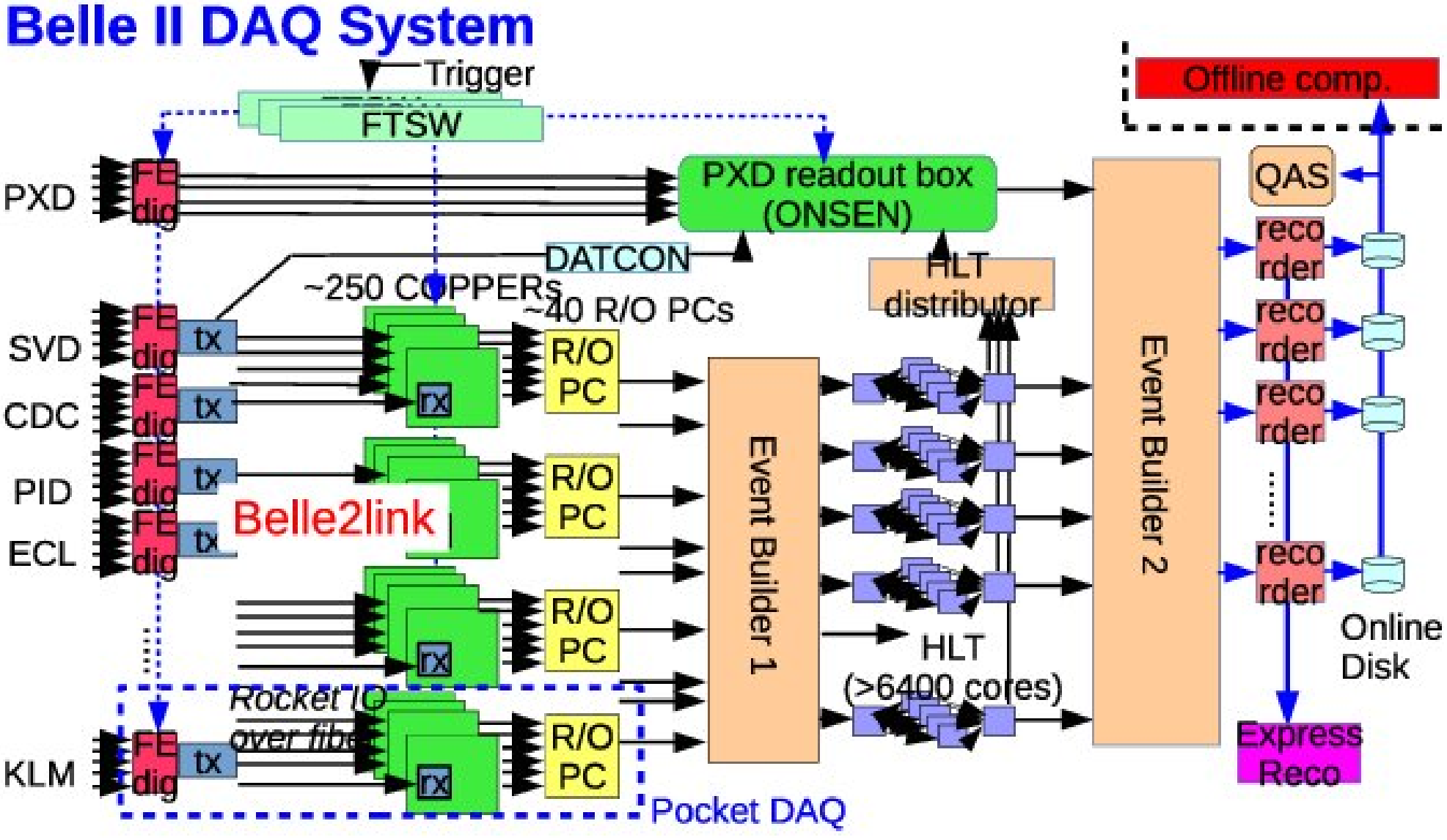
Aman Sahoo
under supervision of Prof. Mikihiko NAKAO

as of July 23, 2025

Belle II Subdetectors

7 Major Subdetectors:

- **PXD** – Pixel Detector: Vertex detection close to interaction point
- **SVD** – Silicon Vertex Detector: Precise tracking of charged particles
- **CDC** – Central Drift Chamber: Momentum measurement and tracking
- **TOP** – Time-of-Propagation Counter: Particle identification (barrel region)
- **ARICH** – Aerogel Ring-Imaging Cherenkov: Particle ID (forward endcap)
- **ECL** – Electromagnetic Calorimeter: Measures energy of photons/electrons
- **KLM** – K_L^0 and Muon Detector: Muon detection and K_L meson ID


Belle II DAQ System
Trigger
FTSW
Offline comp.
PXD
FE dig
PXD readout box (ONSEN)
QAS
DATCON
~250 COPPERs
~40 R/O PCs
HLT distributor
SVD
CDC
PID
ECL
tx
rx
R/O PC
Belle2link
Event Builder 1
Event Builder 2
reco rder
HLT (>6400 cores)
Online Disk
Rocket IO over fiber
KLM
Express Reco
Pocket DAQ

Trigger and Timing System Overview

Level-1 Trigger (L1):

- Hardware-based trigger system
- Reduces event rate to 30 kHz

Need for Synchronization:

- Event-level correlation across subsystems
- Accurate distribution of trigger + time info

Trigger Timing Distribution (TDD):

- System-wide clock to FEE, COPPER, trigger
- Run number, trigger type, event number, timestamp to FEE
- Injection veto and revolution signal broadcast

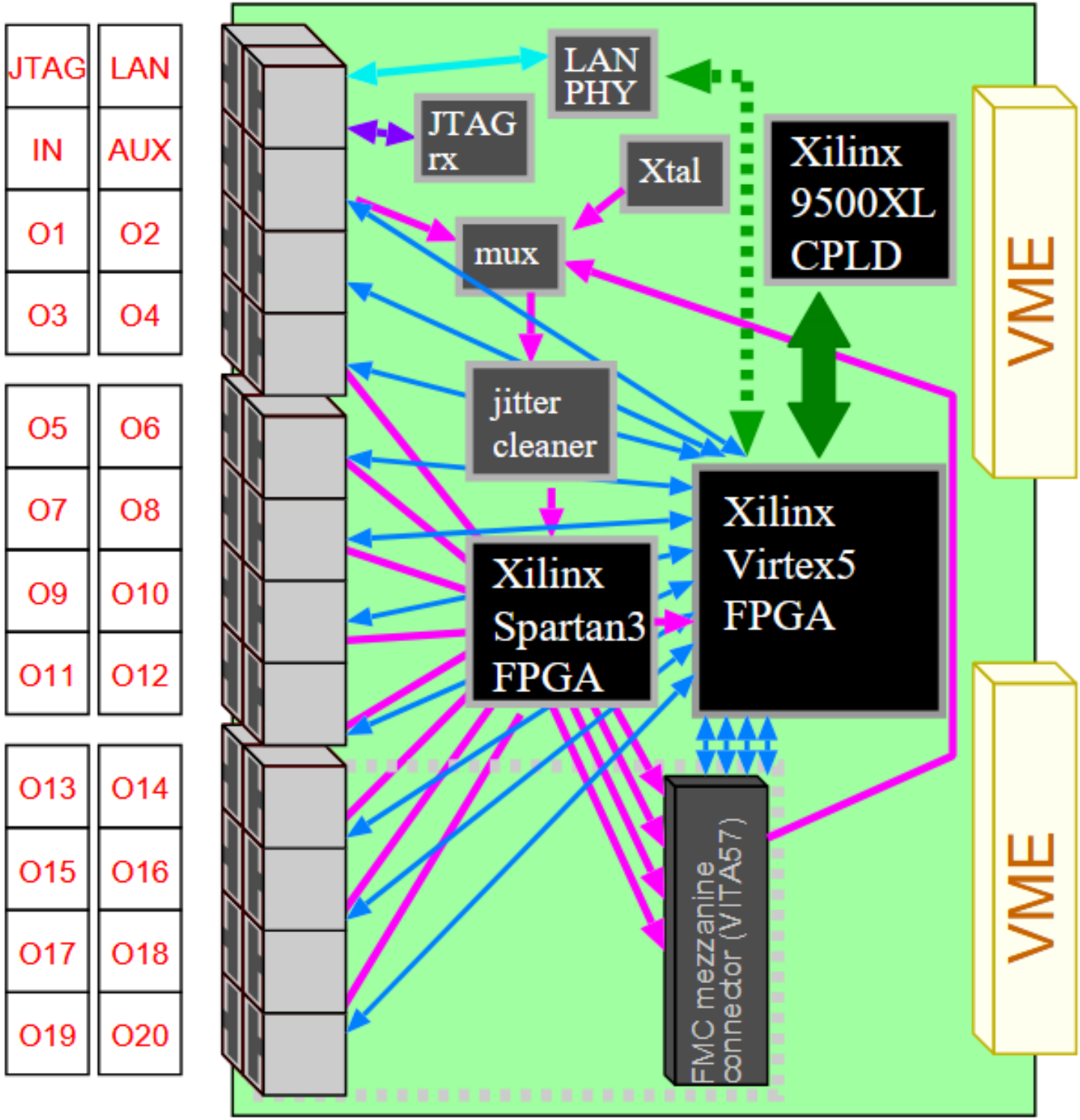
Clocking and FTSW Boards

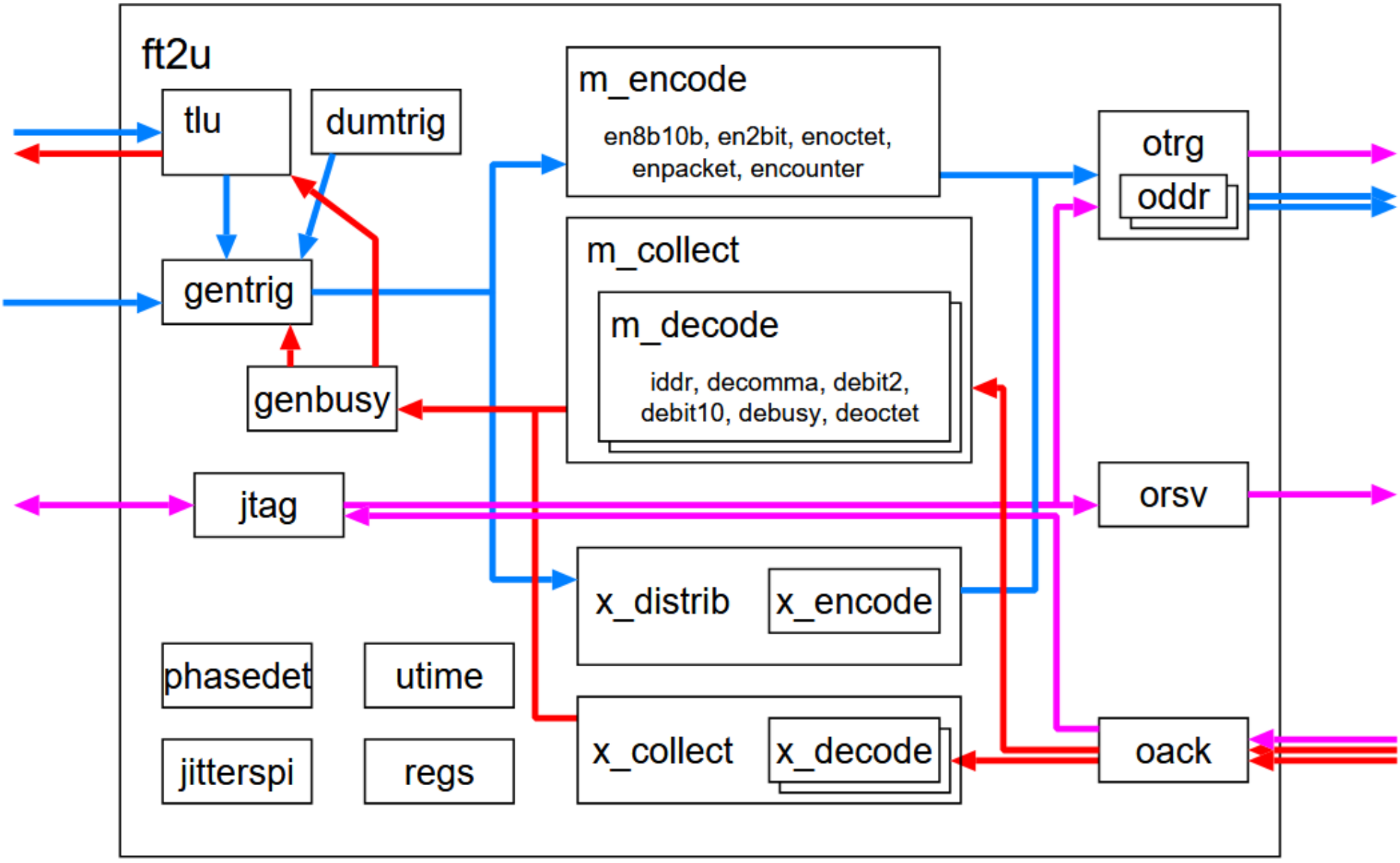
Clock Synchronization:

- SuperKEKB RF clock at 508.9 MHz
- Divided by 4 $\rightarrow$ 127.16 MHz (used by FTSW)

FTSW – Frontend Timing Switch:

- FPGA-based boards
- Aggregate FEE data + trigger + timing
- Transmit using **b2tt protocol**


JTAG
LAN
IN
AUX
O1
O2
O3
O4
O5
O6
O7
O8
O9
O10
O11
O12
O13
O14
O15
O16
O17
O18
O19
O20
LAN PHY
JTAG rx
Xtal
mux
Xilinx 9500XL CPLD
VME
jitter cleaner
Xilinx Virtex5 FPGA
Xilinx Spartan3 FPGA
FMC mezzanine connector (VITA57)
VME


ft2u
tlu
dumtrig
gentrig
genbusy
jtag
phasedet
utime
jitterspi
regs
m_encode
en8b10b, en2bit, enoctet,
enpacket, encounter
m_collect
m_decode
iddr, decomma, debit2,
debit10, debusy, deoctet
x_distrib
x_encode
x_collect
x_decode
otrg
oddr
orsv
oack

01-led (08/07/2025)

- No pattern executed; all LEDs were on simultaneously (static).
- **Edit (10/07/2025):** Code did not function correctly. Likely cause was declaring one_sec in the sensitivity list of the process without invoking its check within the process body.

```
proc_second: process (clk_1) -- changes value of one_sec
    every second

begin
  if rising_edge (clk_1) then

    --some code
    one_sec <= not one_sec;
  end if;
end process;

proc_seq: process (one_sec)
```

02-cycle-led (10/07/2025)

- Pattern cycling functionality achieved.
- Arrays were defined and used as Look-Up Tables (LUTs).
- LUTs contained explicitly described patterns and number of cycles per pattern.
- **Downside:** Every pattern and cycle count was manually typed instead of being generated via logical derivation.
- **Next Step:** Streamline the code and explore more sophisticated architectural design.

```
-- Pattern definitions
type pattern_array is array (0 to 3, 0 to 3) of std_logic_vector(3 downto 0);
constant LED_PATTERNS : pattern_array := (
  ("1000", "0100", "0010", "0001"),
  ("1100", "0110", "0011", "1001"),
  ("1110", "0111", "1011", "1101"),
  ("1000", "1100", "1110", "1111")
);

-- Number of cycles for each pattern (0 = 1 cycle, 1 = 2 cycles)
type cycle_array is array (0 to 3) of integer;
constant PATTERN_CYCLES : cycle_array := (0, 1, 0, 1); -- 1, 2, 1, 2 cycles
```

03-led-cycle-proc (10/07/2025)

- Produced nearly identical output as 02 by dividing functionality into two processes:
  - One for counting seconds.
  - One for tracking cycles and outputting LED signals.
- Gained understanding of proper use of rising edge and synchronization principles.
- Plan to split main process into further subsystems to assess modularity benefits.
- **Curiosity:** Each cycle lasts twice as long as specified. Temporarily resolved by halving time, but root cause needs investigation.

Code

```
proc_second: process (clk_l) -- changes value of one_sec every second

begin
  if rising_edge (clk_l) then

    if cnt_lclk = ONE_SEC_31MHZ - 1 then
      cnt_lclk <= (others => '0');
    else
      cnt_lclk <= cnt_lclk + 1;
    end if;

  end if;
end process;

proc_seq: process (clk_l)

begin
  if rising_edge (clk_l) and cnt_lclk = 0 then
  -- lots of nested if-else statements to handle sequence changes
   -- FTSW3 LED is active-low ('0' for ON / '1' for OFF)
    o1ledg_o <= not seq_led(0);
    o3ledg_o <= not seq_led(1);
    o5ledg_o <= not seq_led(2);
    o7ledg_o <= not seq_led(3);

    interval <= interval - 1;

  end if;
```

04-fast-pattern (15/07/2025)

- Developed simple VHDL code to generate signals faster than previous 1-second LED patterns.
- Signals generated every clock cycle (effectively every 4 clock cycles at 127 MHz).
- Utilized oscilloscope for signal visualization.
- Only TRG channel was used for output.
- Pattern: *x cycles up, x cycles down*, where $x \in \{1, 2, 3, 4\}$ and changes every second.
- Learned to use circuit schematics and .NET files to generate .UCF files.
- Improved understanding of:
  - Differential signals
  - `IOBUFDS`
  - Signal manipulation
- **Next Step:** Utilize RSV and ACK channels and design more complex

Code

```
map_o_trg: obufds port map ( i=> trg_out, o => o_trg_p, ob => o_trg_n );

proc_second: process(clk_l)

begin
    -- code to keep count of runtime in seconds
end process proc_second;

proc_pattern: process(clk_l)

begin
    if rising_edge(clk_l) then

        if cnt_lclk = 0 then
            clock_cycle_up <= clock_cycle_up + 1;
        elsif clock_cycle_up = "00" then
            trg_out <= signal_base(0);
        elsif clock_cycle_up = "01" then
            trg_out <= signal_base(1);
        elsif clock_cycle_up = "10" then
            trg_out <= signal_base(2);
        elsif clock_cycle_up = "11" then
            trg_out <= signal_base(3);
        else
            trg_out <= '0'; -- default case
        end if;

        -- Update signal_base for next cycle
```

05-fast-pattern-lut (16/07/2025)

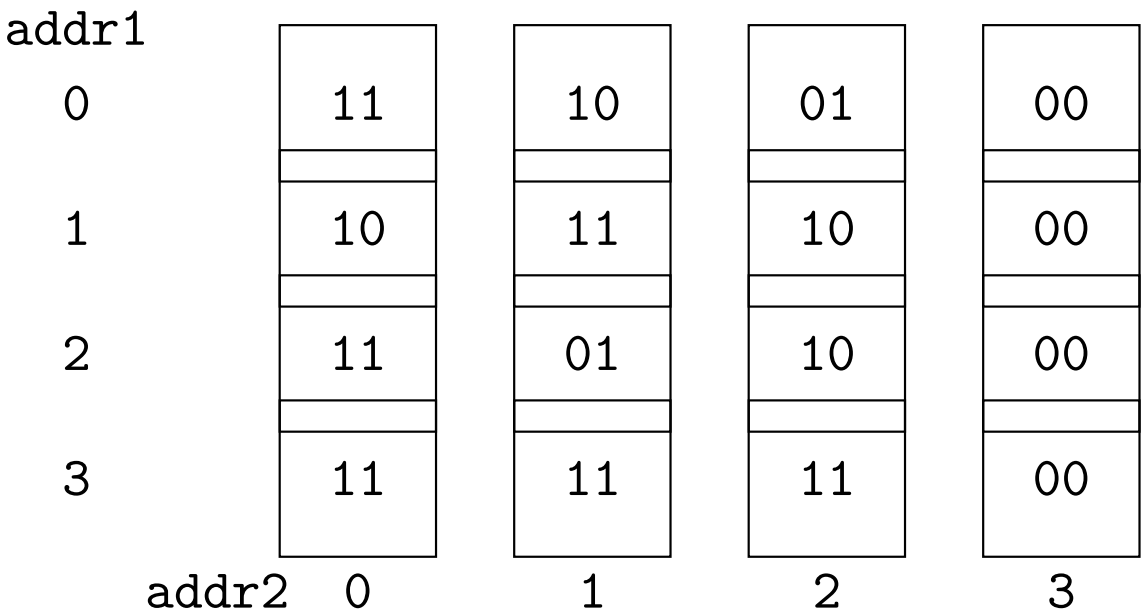
- Extended to include RSV channel alongside TRG for output signal pairs.
- Patterns stored in LUT, updated every second.
- Each pattern starts with a unique “00” signal to indicate sequence start.
- Improved modularity:
  - LUT coded in a separate file: `pattern_lut.vhd`
  - Instantiated in top module via `work` library.
- Yet to use `FOR` loops due to wrap-around behavior of `std_logic_vector`.
- **Next Step:** Test signal functionality on FTSW-3 board.

Code

```
    -- diff sig output - through obufds
    map_o_trg: obufds port map ( i=> trg_out
        , o => o_trg_p, ob => o_trg_n );
    map_o_rsv: obufds port map ( i =>
        rsv_out, o => o_rsv_p, ob =>
        o_rsv_n );

    -- lookup table for pattern generation
    map_lut: entity work.pattern_lut
        port map (
            addr1 => row,
            addr2 => col,
            result => pattern );

    -- processes
    proc_second: process(clk_1)

    begin
-- code to keep time
    end process proc_second;

    proc_pattern: process(clk_1)

    begin
        if rising_edge(clk_1) then
            trg_out <= pattern(0);
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pattern_lut is
    port (
        addr1 : in std_logic_vector(1
            downto 0); -- 2-bit address
             input
        addr2 : in std_logic_vector(1
            downto 0);
        result : out std_logic_vector(1
            downto 0) );
end pattern_lut;

architecture rtl of pattern_lut is

    type lut_type is array (0 to 3, 0 to
         3) of std_logic_vector(1
        downto 0);
    constant lut : lut_type := (
        ( "11", "10", "01", "00" ),
        ( "10", "11", "10", "00" ),
        ( "11", "01", "10", "00" ),
        ( "11", "11", "11", "00" )
    );
begin
```

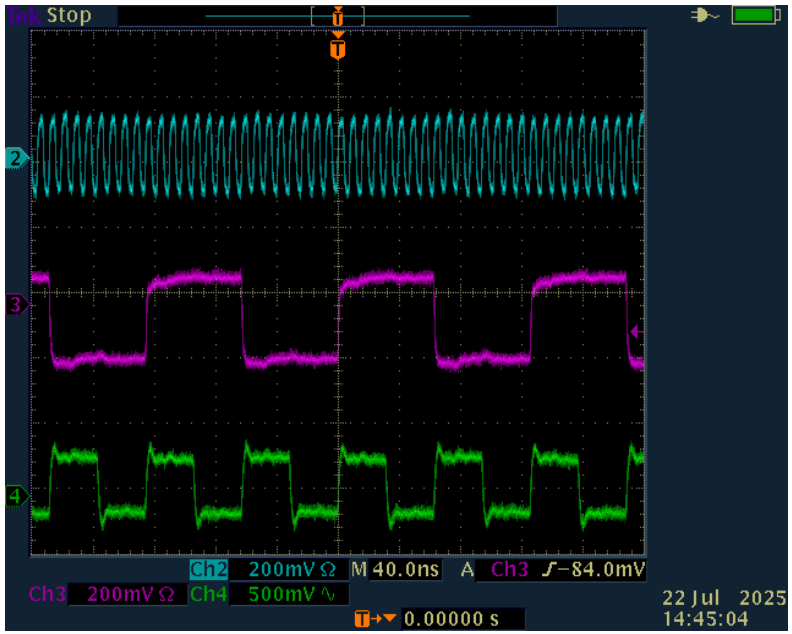
LUT-Based Signal Output Diagram

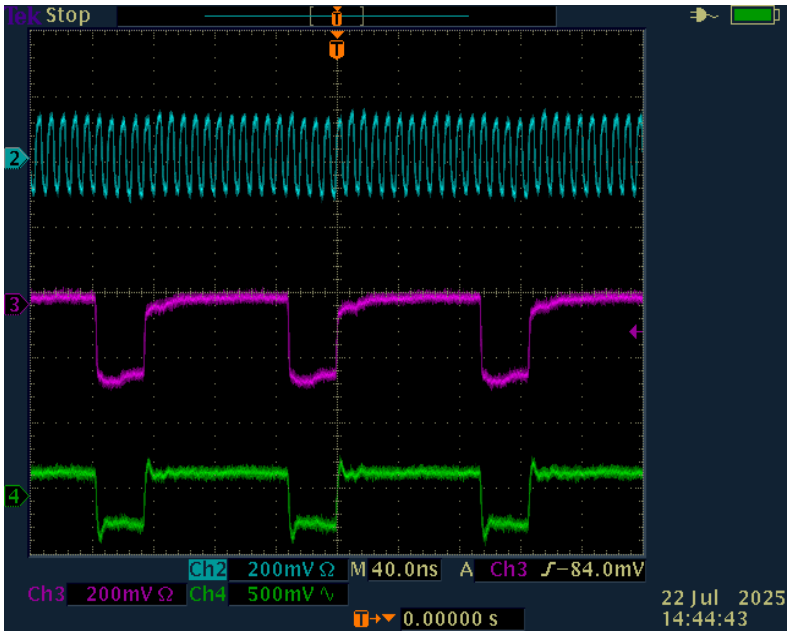


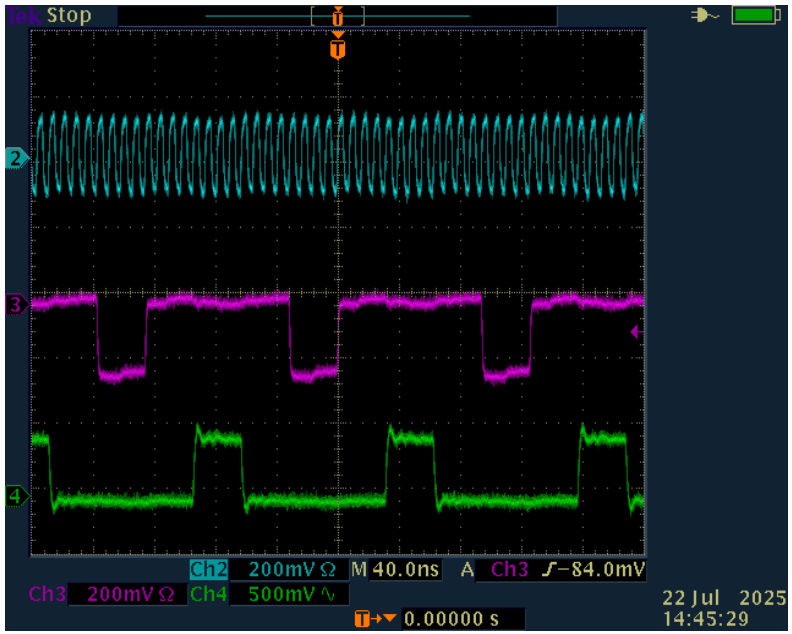
Output:

`result(1) = Signal 1`
`result(0) = Signal 2`


Tek Stop
Ch2 200mV Ω M 40.0ns A Ch3 ∫−84.0mV
Ch3 200mV Ω Ch4 500mV ∿
0.00000 s
22 Jul 2025
14:44:12


Tek Stop
Ch2 200mV Ω M 40.0ns A Ch3 ∫−84.0mV
Ch3 200mV Ω Ch4 500mV ∿
0.00000 s
22 Jul 2025
14:45:04


Tek Stop
Ch2 200mV Ω M 40.0ns A Ch3 ∫−84.0mV
Ch3 200mV Ω Ch4 500mV ∿
0.00000 s
22 Jul 2025
14:44:43


Tek Stop
Ch2 200mV Ω M 40.0ns A Ch3 ∫−84.0mV
Ch3 200mV Ω Ch4 500mV ∿
0.00000 s
22 Jul 2025
14:45:29

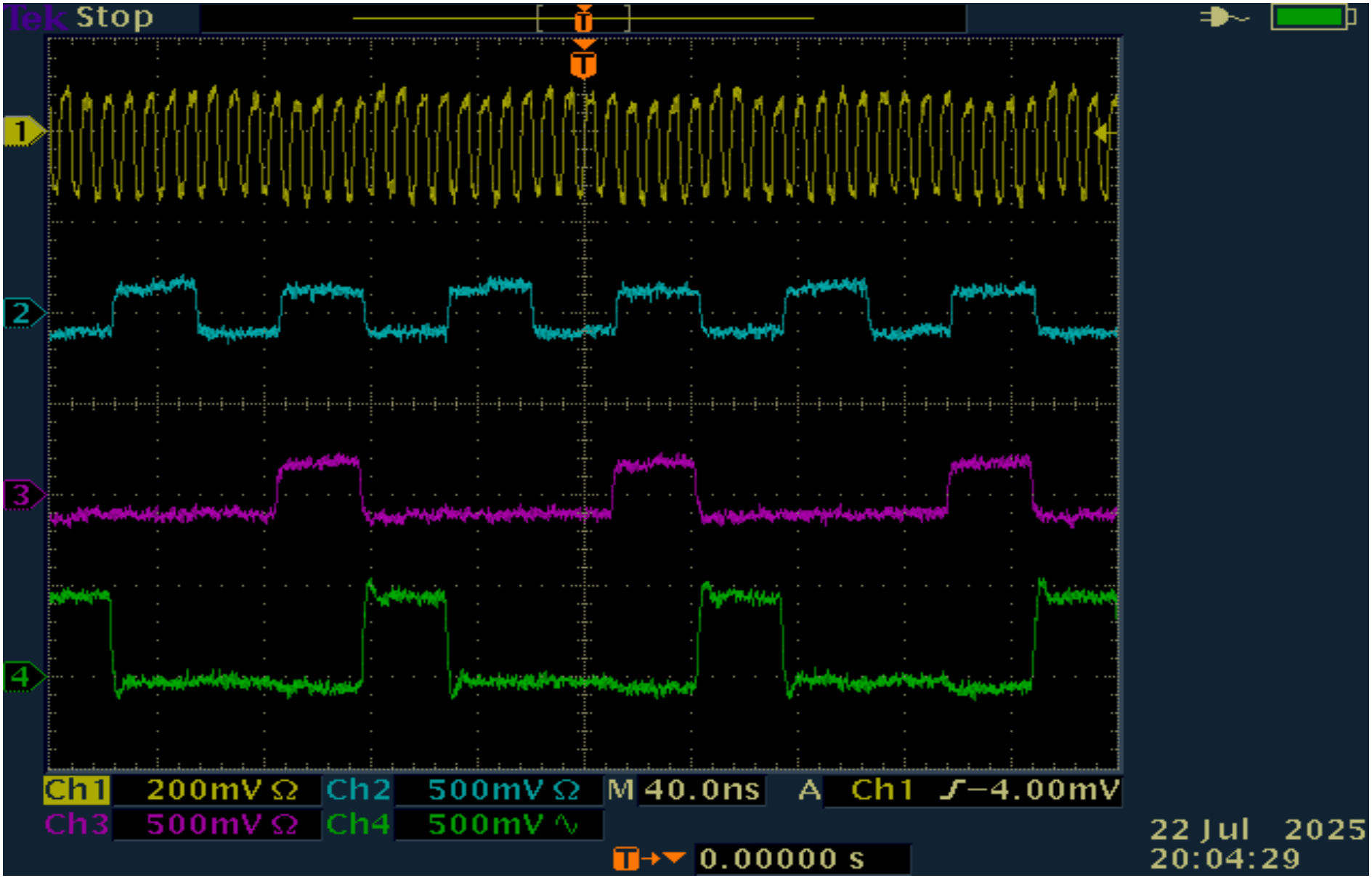
06-signal-testing (22/07/2025)

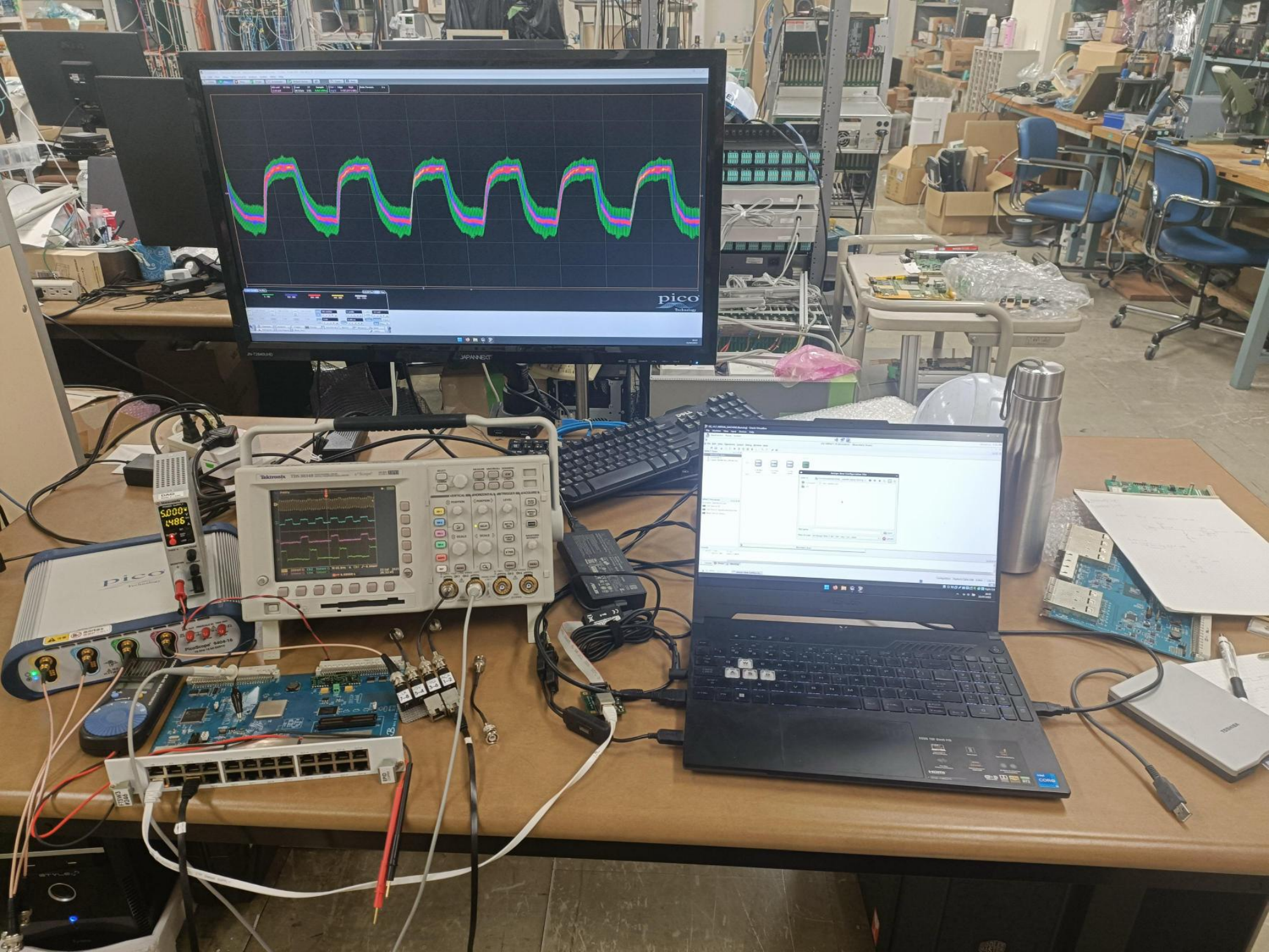
- Based on design from 05-fast-pattern-lut to test all FPGA output/input ports (LAN and AUX excluded).
- All port channels used: CLK, RSV, ACK, TRG.
- Defined three 20-bit vectors to map specific outputs per channel per port.
- Created `obufds_module` and instantiated in top module.
- Purpose: Convert `std_logic_vector` signals to differential output vectors.
- Used `generate` statement to replicate `obufds` for each port.
- Added LED logic: LED pattern shifts every 0.5 seconds for verification.
- **Results:**
  - Verified output from all ports (OUT-1 to OUT-20) using oscilloscope and LEDs.
  - LED, RSV, TRG, ACK channels operate correctly.
  - CLK channel functions on odd-numbered ports, but fails on even-numbered ports.
  - Requires further investigation.

Code

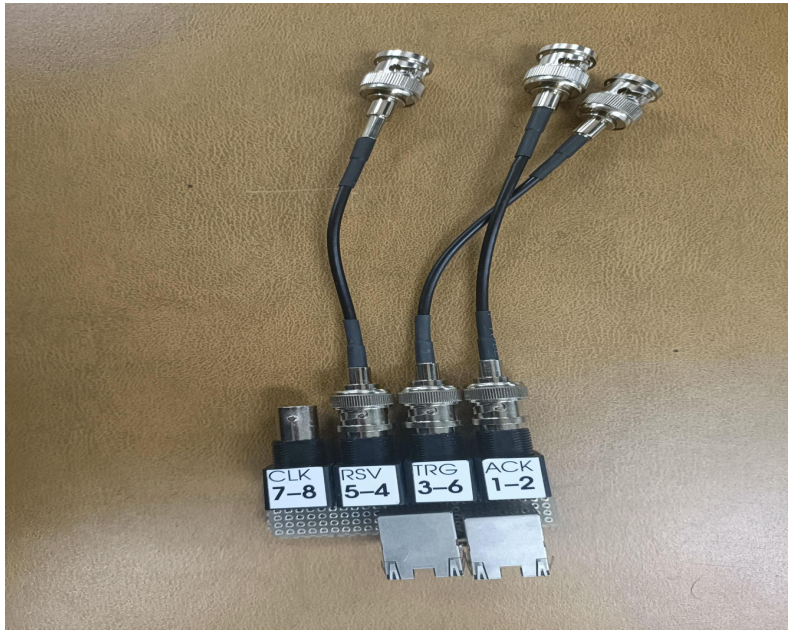
```
proc_pattern: process(clk_l)

begin
    if rising_edge(clk_l) then
        trg_out <= ( others => pattern
            (0) );
        rsv_out <= ( others => pattern
            (1) );
        ack_out <= ( others => pattern
            (2) );
        col <= col + 1;
    end if;
end process proc_pattern;

proc_led: process(clk_l)

begin
    if rising_edge(clk_l) then
        if cnt_lclk = (ONE_SEC_31MHZ
            /2) - 1 or cnt_lclk =
            ONE_SEC_31MHZ - 1 then
            led <= not led;
        end if;

        o_ledy_b <= (others => led);
        o_ledg_b <= (others => not led
            );
```

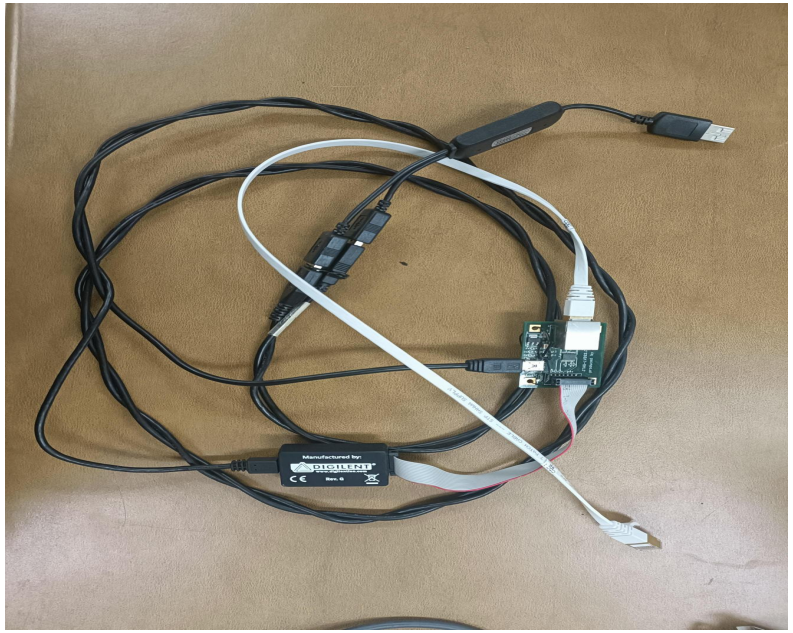
```
entity obufds_module is
    port (
        trg_out : in std_logic_vector (20
            downto 1);
        o_trg_n : out std_logic_vector (20
            downto 1);
        o_trg_p : out std_logic_vector (20
            downto 1)
        -- similar ports for rsv and ack
        );
end obufds_module;

architecture rtl of obufds_module is

begin
    GEN_OBUFDS: for x in 1 to 20 generate
        map_o_trg: obufds port map ( i =>
            trg_out(x), o => o_trg_p(x),
            ob => o_trg_n(x) );
        map_o_rsv: obufds port map ( i =>
            rsv_out(x), o => o_rsv_p(x),
            ob => o_rsv_n(x) );
        map_o_ack: obufds port map ( i =>
            ack_out(x), o => o_ack_p(x),
            ob => o_ack_n(x) );
    end generate GEN_OBUFDS;
end rtl;
```

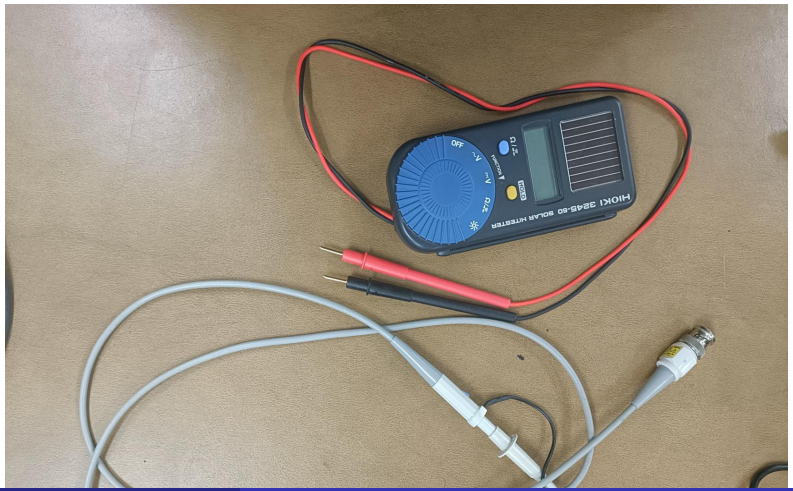

Tek Stop
1
2
3
4
Ch1 200mV Ω
Ch2 500mV Ω
M 40.0ns
A Ch1 ∫ −4.00mV
Ch3 500mV Ω
Ch4 500mV ∿
0.00000 s
22 Jul 2025
20:04:29


pico
Tektronix
pico

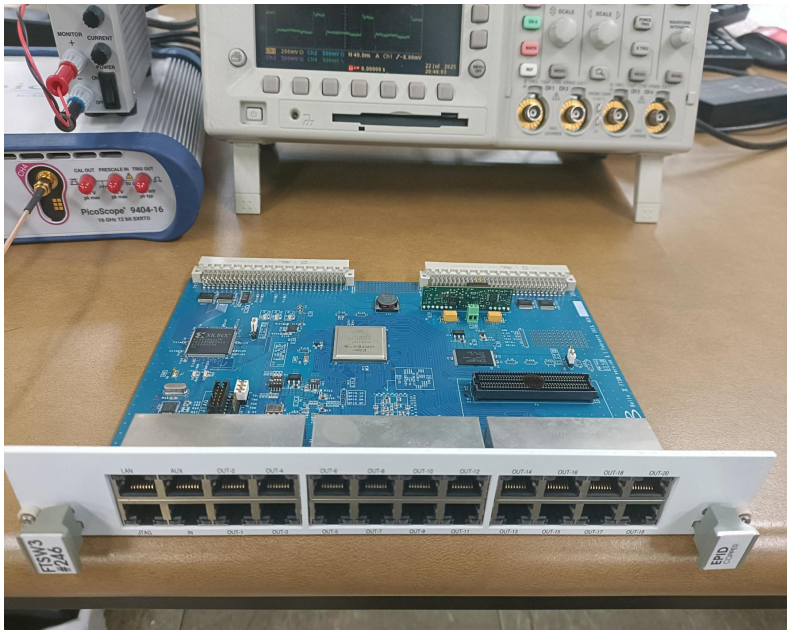
Cable Equipment

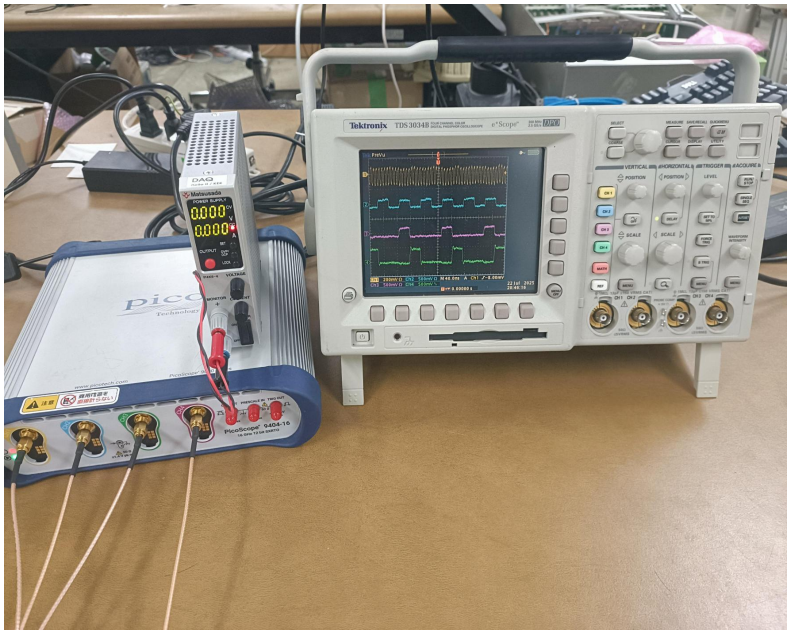






Cable Equipment






Clear
Run
Stop
Single
Autoscale..
Default Setup..
Copy..
Help
100 mV/ 16 GHz
-3.14 mV
5 ns/ ET Sample
40 GSa/s 2 KS 15.76 kWfm
Ch4 / Edge High
Trig'd f=127.217 2 MHz
Dots Persist. 2 s
C4
T
Color Grade
Scales
Auto
Max
Mid
Min
1 - 185
186 - 371
372 - 742
743 - 1484
1485 - 2969
pico
Technology
Ch1
100 mV/div
Ch2
100 mV/div
Ch3
100 mV/div
Ch4
100 mV/div
0 div
5 ns/div
5.325 ns
-3.1 mV
Freerun
Trig'd
Pos
Neg
Bi
Channels
Acquire
Trigger
Display
Save/Recall
Marker
Measure
Math
Histogram
Eye Diagram
Mask Test
Utility

Thank You!

Questions and Discussion Welcome