

# 2026年2月ブログ集

## 目次

2026年02月01日	北房のブリ市.....	2
2026年02月02日	農薬の在庫調査.....	2
2026年02月03日	今週末は衆議院議員選挙だけど・・・.....	2
2026年02月04日	仮剪定を開始.....	2
2026年02月05日	MicroPython のSPI ライブラリでの失敗談.....	3
2026年02月06日	旧建部町方面にお出かけ.....	8
2026年02月07日	ShrikeLite AI に聞いて試した5分周回路.....	9
2026年02月08日	AI に周波数カウンタを設計してもらおう.....	10
2026年02月09日	好機到来.....	14
2026年02月10日	倉敷総社方面にお出かけ.....	15
2026年02月11日	2025 CQ WW contest (SSB)の結果発表.....	15
2026年02月12日	イチジクなどの剪定.....	15
2026年02月13日	ブドウの本剪定を開始.....	16
2026年02月14日	AU7RS LAKSHADWEEP ISLANDS.....	16
2026年02月15日	Shrike Lite で周波数カウンタを作った.....	16
2026年02月16日	AI で2列2段八木のNECファイルを作った.....	30
2026年02月17日	令和7年度確定申告.....	30
2026年02月18日	ブドウの剪定作業完了.....	31
2026年02月19日	TANG NANO 9K FPGA BOARD.....	31
2026年02月20日	KP5/NP3VI on 10m.....	31
2026年02月21日	ため池の水位.....	31
2026年02月22日	剪定枝を焼却.....	32
2026年02月23日	キウイフルーツの剪定.....	32
2026年02月24日	TangNano9k でLチカ初トライ.....	32
2026年02月25日	Where is the 3Y0K team now?.....	33
2026年02月26日	農大の親睦会.....	33
2026年02月27日	杭の製作.....	34
2026年02月28日	フラクタルが美しい野菜.....	34

## 2026年02月01日 北房のブリ市

今日は、家内と娘婿夫婦および孫達と一緒に、北房のブリ市に出かけました。京都で水揚げされた体長約85cmの天然物を半身買いました。婿殿は釣りをするだけあっても、魚を捌いて料理する事も得意としているので、お昼は早速料理してもらってごちそうになりました。ブリカマの焼き物に、ブリしゃぶというメニューに舌鼓を打ちました。アラの煮付けと、切り身をお土産として頂きました。ブリ三昧です。

## 2026年02月02日 農薬の在庫調査

2月9日までに農薬の予約注文書を提出する必要があるので、農薬の在庫を調査しました。なるべく農薬が残らないように、小さな瓶や小袋の物を購入するように心掛けていますが、中には500g単位でしか注文できないような農薬もありますし、どうしても端数が残ってしまいます。また農薬には使用期限があるので、先入先出で、期限の切れそうなものから順に使っていく必要があります。いずれにしても帳簿で農薬の管理をすることは重要です。

## 2026年02月03日 今週末は衆議院議員選挙だけど・・・

今週末の2月8日には衆議院議員選挙があります。まだ誰に投票するか決めていません。先日、こんな田舎に選挙カーが回ってきましたが、見慣れない（聞きなれない）候補者だったので、アレっ？と感じました。まあ近頃は女性の候補者も多いので、当たり前のことかもしれませんが、自民党の経験豊富な候補者でした。今回の選挙では、投票用紙は郵送されたものの、選挙公報等は配布されていないので、ちょっと調べてみました。すると、大変なことがわかりました。なんと、選挙区が大幅に改定されていたのです。この新しい選挙区での総選挙は、今回で2回目です。変更された区割りが施行されたのが、令和4年12月28日で、前回の総選挙は令和6年10月27日だったのです。前回の総選挙の時には何とも思いませんでしたが、誰に投票したのか記憶にないくらいなので仕方ありません。

## 2026年02月04日 仮剪定を開始

今日は、立春の名にふさわしい穏やかな良い天気でした。午前中はウォーミングアップのために薪割りをしました。午後から暖かくなってきたので、久々にぶどう園に行き

ました。仮剪定は立春が過ぎてか始めようと思っていたので、満を持しての作業開始です。午後4時頃まで作業して、約50%完了しました。但し、支線の辺りで切っただけで、防鳥ネットに引っ掛かっている枝もあるため、それらを取り外すのに一手間掛ります。このため仮選定の作業は、あと2日程かかりそうです。

## 2026年02月05日 MicroPythonのSPIライブラリでの失敗談

ShrikeLite という RP2040+FPGA が搭載されたボードで遊んでいます。MicroPython を RP2040 側で動作させて、FPGA 側に組み込んだ SPI のデバッグをしている時の話です。SPI を経由して FPGA 側から 32 ビットのデータが読み出せるようにしたかったので、FPGA 側ではデータ幅 32 ビットの SPI を作り込みました。シミュレータで動作確認したところ良い感じに動作していましたが、RP2040 側でデータを読み出すと、0x01234567 という固定データが読み出される筈なのに、0x23672367 と読み出されるのです。何でだろう？と思って、とうとうオシロスコープまで接続して波形を確認すると、何と SCK が 32 でも 8 でもなく、16 ずつ出ているではありませんか！これが冒頭の画像です。MSBfirst なので、データは 0x0123 と読取る事ができます。

ネットをググったり AI に聞くと、そもそも Arduino の SPI ライブラリは 8 ビットのリードライトにしか対応していないというのです。しかし、10 年程前に ESP32 と CycloneII の組み合わせで GPSDO を製作した時には、SPI.transfer32 というメソッドを使っていたので、てっきり SPI で 32 ビットデータのやり取りはできるものだと思い込んでいました。ところが、RP2040 の earlphilhower の SPI ライブラリには SPI.transfer32 というメソッドは見当たりませんでした。MicroPython はどうかと言うと、spi.write\_readinto の例では 8 バイトデータを送受信する例が示されているので、32 ビット (4 バイト) のデータ転送もできるだろうと思ったのが運の尽きでした。何故一度に 16 ビットのデータをバースト転送するのか理解に苦しみましたが、色々やってみたところ、次に示す SPI コンフィグレーションに問題があることが分かりました。

```
spi = SPI(0,  
          baudrate=100_000,  
          polarity=0,  
          phase=0,  
          bits=32,
```

```
firstbit=SPI.MSB,  
sck=Pin(SCK),  
mosi=Pin(MOSI),  
miso=Pin(MISO))
```

この中の `bits=32` という記述が悪さをしていたのです。 `bits=8` と書き換えて `spi.read` メソッドで4バイトのデータを受信するように変更すると、期待どおりに動作するようになりました。

```
MPY: soft reboot  
[shrike_flash] FPGA reset done  
[shrike_fpga] Starting FPGA flash...  
[shrike_fpga] flashing: SPI32const.bin  
[shrike_flash] FPGA programming done.  
Received 00, 00, 00, 00  
Received 01, 23, 45, 67  
Received 01, 23, 45, 67  
Received 01, 23, 45, 67  
Received 01, 23, 45, 67
```

この時の MicroPython のソースコードは次の通りです。この例では受信データは `bytearray` 型として記述していますが、32ビットデータに変換する際にはエンディアンにも注意する必要があります。

```
import shrike  
import machine  
from machine import Pin, SPI  
import time  
shrike.flash("SPI32const.bin")  
  
# Reset  
reset_pin = Pin(14, Pin.OUT, value=1)  
reset_pin.value(0)  
time.sleep(1)  
reset_pin.value(1)  
time.sleep(1)  
  
# RP2040
```

```

SCK = 2
CS = 1
MOSI = 3
MISO = 0

# Chip Select pin
cs = Pin(CS, Pin.OUT, value=1)

# SPI configuration (MODE 0, MSB first)
spi = SPI(0,
    baudrate=100_000,
    polarity=0,
    phase=0,
    bits=8,
    firstbit=SPI.MSB,
    sck=Pin(SCK),
    mosi=Pin(MOSI),
    miso=Pin(MISO))

def spi_exchange32(byte_to_send):
    tx = bytearray(4)
    rx = bytearray(4)
    tx[3] = byte_to_send
    rx = spi.read(4,byte_to_send)
    print(f"Received {rx[0]:02x}, {rx[1]:02x}, {rx[2]:02x}, {rx[3]:02x}")

while True:
    for val in [0xAA, 0xFF]:
        cs.value(0)      # Select FPGA
        resp = spi_exchange32(val)
        cs.value(1)      # Select FPGA
        time.sleep(1)

```

ついでに、FPGA 側の Verilog によるソースコードを以下に示します。  
 spi32\_target.v は、ForgeFPGAworkshop のライブラリなので、割愛します。デバック（波形観測）のために、SCK と MISO 信号を別のピンに出力するようにしています。

```

(* top *) module SPI_const (
    (* iopad_external_pin, clkbuf_inhibit *) input clk,      // System Clock
    (50MHz)
    (* iopad_external_pin *) output clk_en,
    (* iopad_external_pin *) input rst_n,                  // System
Reset (Active Low)

    // Physical SPI Pins (Connect these to FPGA I/O)
    (* iopad_external_pin *) input spi_ss_n,
    (* iopad_external_pin *) input spi_sck,
    (* iopad_external_pin *) input spi_mosi,
    (* iopad_external_pin *) output spi_miso,
    (* iopad_external_pin *) output spi_miso_en,

    (* iopad_external_pin *) output reg debug1,
    (* iopad_external_pin *) output debug1_en,
    (* iopad_external_pin *) output reg debug2,
    (* iopad_external_pin *) output debug2_en,

    // Physical LED Pins
    (* iopad_external_pin *) output reg led,
    (* iopad_external_pin *) output led_en
);

assign debug1_en = 1'b1;
assign debug2_en = 1'b1;
always @(posedge clk) begin
    if (spi_sck)
        debug1 <= 1'b1;
    else
        debug1 <= 1'b0;
end

always @(posedge clk) begin
    if (spi_miso)
        debug2 <= 1'b1;

```

```

        else
            debug2 <= 1'b0;
        end
        assign led_en = 1'b1;
        assign clk_en = 1'b1;

wire [31:0] rx_data_wire;
wire      rx_valid_pulse;
reg  [31:0] tx_data_reg;
wire tx_hold_pulse;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        tx_data_reg <= 32'h0000_0000;
    end else if (tx_hold_pulse) begin
        tx_data_reg <= 32'h0123_4567;
    end
end

// LED Logic
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        led <= 1'b0;
    end else if (rx_valid_pulse) begin
        if (rx_data_wire[0] == 1'b1)
            led <= 1'b1;
        else
            led <= 1'b0;
    end
end

// SPI Target
spi32_target #(
    .CPOL(1'b0), // Standard Mode 0 (Idle Low)
    .CPHA(1'b0), // Standard Mode 0 (Sample Rising)
    .WIDTH(32),

```

```

        .LSB(1'b0)    // MSB First (Standard)
) u_spi_target (
    // System Common
    .i_clk(clk),
    .i_rst_n(rst_n),
    .i_enable(1'b1),    // Enable the module permanently

    // SPI Physical Interface
    .i_ss_n(spi_ss_n),
    .i_sck(debug1),
    .i_mosi(spi_mosi),
    .o_miso(spi_miso),
    .o_miso_oe(spi_miso_en),

    // RX Interface (Data FROM MCU)
    .o_rx_data(rx_data_wire),
    .o_rx_data_valid(rx_valid_pulse),

    // TX Interface (Data TO MCU)
    .i_tx_data(tx_data_reg),
    .o_tx_data_hold(tx_hold_pulse)    // Not needed for simple echo
);

endmodule

```

## 2026年02月06日 旧建部町方面にお出かけ

警察署に用事があったので、天気は良さそうでしたが、野良仕事は無しにして、家内と一緒に出かけました。まずは、農協に寄って、農薬や農業資材の年間注文書を出しました。お次は、北警察署に出向いて、銃所持許可証の更新をしました。其処から約15分、すぎ茶屋に開店直後の11時過ぎに到着しました。私の注文は、いつもの「名物鍋焼きうどん」です。具沢山で本当に美味しい！お腹を満たした後、八幡温泉で小一時間程過ごしました。帰りにマルナカに寄って買い物をして、2時半頃帰宅しました。温泉やサウナで汗を流すと、心は休まっても体は消耗するのか眠くなったので、帰宅後はお昼寝しました。今日は良い骨休みができました。

## 2026年02月07日 ShrikeLite AIに聞いて試した5分周回路

AI (Gemini) に次のような質問を投げかけて得られた回答を元に、ShrikeLiteで5分周回路を試してみました。質問「ステートマシンを利用してシステムクロックを5分周する回路をverilogで記述してください」

Geminiの回答をForgeFPGAworkshopのmain.vの下の方に別のmoduleとしてコピペし、topモジュール内で次のように記述しました。Verilogの文法は勉強中ですが、AIに教えてもらいながら何とかやりたいことはできました。

```
//divider to generate 10MHz
clock_divider_5 divid5(
    .clk(clk),
    .reset_n(rst_n),
    .clk_out(clk10MHz)
);
reg clk10MHz;
assign debug1 = clk10MHz;
```

Geminiの回答では、デューティ比が1:5になるので、次のように変更して2:5にしました。冒頭の画像は、その出力波形です。デジタルオシロの表示では周波数は10.09MHz前後でした。

```
case (state)
    S0: begin
        state <= S1;
        clk_out <= 1'b1; // S0のタイミングでHighを出力
    end
    S1: begin
        state <= S2;
        clk_out <= 1'b1; // same as S1
    end
    S2: begin
        state <= S3;
        clk_out <= 1'b0;
    end
```

```

S3: begin
    state <= S4;
    clk_out <= 1'b0;
end
S4: begin
    state <= S0;
    clk_out <= 1'b0;
end
default: begin
    state <= S0;
    clk_out <= 1'b0;
end
endcase

```

## 2026年02月08日 AIに周波数カウンタを設計してもらう

ShrikeLiteで周波数カウンタを製作したいと思っています。過去にGPSDOを製作したことがあります。10MHzで10ppbクラスの周波数確度を目指すのであれば、ゲートを10秒間開けて100Mカウントすることになるので、28ビット程度の桁数が必要です。GPSレシーバの1pps信号を利用してゲートを制御すれば、正確な周波数カウンタを自作することができます。私は、未だVerilogの学習途上なので、フルスクラッチで記述する自信がありません。そこで、GPSDOの一部としての周波数カウンタをAIに設計してもらいました。Geminiに次のように問い合わせました。

28ビットの周波数カウンタをverilogで書いてください。

ただし、ゲート時間は、外部から入力されるgate\_inにより制御され、カウント中はbusy信号を出力することとします。

すると、次のようなVerilogのコードを提示されました。

```

module freq_counter_28bit (
    input wire    clk,    // システムクロック
    input wire    reset_n, // 非同期リセット (負論理)
    input wire    sig_in, // 測定対象の入力信号

```

```

input wire    gate_in, // 外部ゲート入力
output reg [27:0] freq_out, // 測定された周波数データ
output reg    busy    // カウント中フラグ
);

// 内部レジスタ
reg [27:0] count_reg;
reg    gate_sync_0, gate_sync_1; // gate_in 同期化用
reg    sig_sync_0, sig_sync_1;  // sig_in エッジ検出用

// 信号の同期化とエッジ検出用バッファ
always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        gate_sync_0 <= 1'b0;
        gate_sync_1 <= 1'b0;
        sig_sync_0  <= 1'b0;
        sig_sync_1  <= 1'b0;
    end else begin
        gate_sync_0 <= gate_in;
        gate_sync_1 <= gate_sync_0;
        sig_sync_0  <= sig_in;
        sig_sync_1  <= sig_sync_0;
    end
end

// エッジ検出フラグ
wire gate_rising  = (gate_sync_0 && !gate_sync_1);
wire gate_falling = (!gate_sync_0 && gate_sync_1);
wire sig_rising   = (sig_sync_0  && !sig_sync_1);

// メイン制御ロジック
always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        count_reg <= 28'd0;
        freq_out  <= 28'd0;
        busy      <= 1'b0;
    end else begin
        if (gate_rising) begin

```

```

        // カウント開始
        count_reg <= 28'd0;
        busy      <= 1'b1;
    end
    else if (busy) begin
        if (gate_falling) begin
            // カウント終了、結果を保持
            freq_out <= count_reg;
            busy      <= 1'b0;
        end else if (sig_rising) begin
            // ゲート期間中の信号立ち上がりをカウント
            count_reg <= count_reg + 1'b1;
        end
    end
end
end
end
end

```

```
endmodule
```

日本語のコメント付きなので、初心者にとっては勉強になります。Gemini は、テストベンチの作成もお手伝いしましょうかとのオファーをくれたので、お願いしたところ、次のようなテストベンチの Verilog コードを提示してくれました。

```
`timescale 1ns / 1ps
```

```
module freq_counter_tb();
```

```
    // 信号定義
```

```
    reg      clk;
```

```
    reg      reset_n;
```

```
    reg      sig_in;
```

```
    reg      gate_in;
```

```
    wire [27:0] freq_out;
```

```
    wire      busy;
```

```
    // クロック周期の定義
```

```
    parameter CLK_PERIOD = 10; // 100MHz
```

```
    parameter SIG_PERIOD = 40; // 25MHz (テスト対象信号)
```

```

// インスタンス化
freq_counter_28bit uut (
    .clk(clk),
    .reset_n(reset_n),
    .sig_in(sig_in),
    .gate_in(gate_in),
    .freq_out(freq_out),
    .busy(busy)
);

// システムクロック生成
initial clk = 0;
always #(CLK_PERIOD/2) clk = ~clk;

// テスト対象信号 (sig_in) 生成
initial sig_in = 0;
always #(SIG_PERIOD/2) sig_in = ~sig_in;

// テストシナリオ
initial begin
    // 初期状態
    reset_n = 0;
    gate_in = 0;

    // リセット解除
    #(CLK_PERIOD * 5);
    reset_n = 1;
    #(CLK_PERIOD * 5);

    // --- 測定サイクル 1 ---
    $display("Measurement 1 start...");
    gate_in = 1;
    #(SIG_PERIOD * 10.5); // sig_in を 10 回分カウントする程度の時間
    gate_in = 0;

    // busy が下がるのを待つ
    wait(busy == 0);

```

```

$display("Result 1: %d pulses counted", freq_out);

#(CLK_PERIOD * 10);

// --- 測定サイクル 2 ---
$display("Measurement 2 start...");
gate_in = 1;
#(SIG_PERIOD * 25.5); // sig_in を 25 回分カウントする程度の時間
gate_in = 0;

wait(busy == 0);
$display("Result 2: %d pulses counted", freq_out);

#(CLK_PERIOD * 20);
$finish;
end

// 波形出力用 (GTKWave 等で確認する場合)
initial begin
    $dumpfile("freq_counter_tb.vcd");
    $dumpvars(0, freq_counter_tb);
end

endmodule

```

この2つのVerilogコードを icarus(iverilog)でコンパイルして GTKWave で確認してみました。Verilog 初心者にとっては、回路設計だけでも大変なのに、シミュレータ用のコード（テストベンチ）まで作ってくれて大変助かります。しかもオールフリー（全部無料）です。私が現役だった 20 年前に比べると夢のような話です。

## 2026 年 02 月 09 日 好機到来

週末は今季一番の寒気に包まれて、昨日は吹雪でした。今朝、起きてみると雪が 10cm くらい積もっていました。天気予報を見ると午前中は風が弱いとのことだったので、木の枝を燃やす作業をしました。こんな朝は焚き火をするには絶好のチャンスです。特に葉っぱのついた広葉樹の枝を燃やすと、火の粉となって周囲に燃え広がる可能

性があります。その点、雪が積もっている朝は何の心配もありません。こんな日をずっと待っていたのです。この日のために、刈って乾燥した草を束ねて 屋根のあるところに焚き着けとして保管していました。1 時間程で綺麗に燃やす事が出来てスッキリしました。

## 2026 年 02 月 10 日 倉敷総社方面にお出かけ

天気予報によると、今日も寒い日でしかも曇りの予報だったので、妻と一緒に倉敷・総社方面にお出かけしました。近所の路上には未だ雪が残っているところがありましたが、町外に出ると雪が降ったことが嘘のような感じでした。今日は倉敷のサイゼリアでランチをして、総社のサントピアで温泉に浸かるというコースでした。帰宅すると 2 時半を回っていたので、コタツで暖を取りながらお昼寝です。こんな日もありませぬ！

## 2026 年 02 月 11 日 2025 CQ WW contest (SSB)の結果発表

先頃、昨年 10 月末に開催された CQ World Wide DX contest SSB の結果が発表されました。私は、2024 年に続き、Single OP, 10m band, Assisted, High-Power として参加しました。2024 年は 15.9 時間運用して、404Q/30Z/80C で、スコアは 122,100、World97 位、アジア 11 位でしたが、JA4 エリアで 2 位という結果でした。今年は 12.1 時間運用しただけでしたが JA4 エリア Top という結果だったのは、偶々運が良かっただけです。運用時間が短い分 QSO 数は少なく 241Q/28Z/82C で、スコアは 73,480 とかなり低めで、当然ながら World 172 位、アジア 25 位と順位を下げました。突発性難聴を患ってから、SSB は特に不得意種目になり、ナンチャテ参加なので、順位に拘るつもりは毛頭ありません。

## 2026 年 02 月 12 日 イチジクなどの剪定

今日は晴れの天気予報でしたが 午前中は今一つだったので、家の近くに植えてあるイチジク、サクランボ(暖地桜桃)、梅の剪定をしました。イチジクは、天敵であるカミキリにかなりヤラれている樹もあり、芽が出るかどうか心配です。カミキリはなかなか駆除できないので改植するのが手っ取り早いのかなぁと思ったりします。

## 2026年02月13日 ブドウの本剪定を開始

寒い日が一週間ばかり続いていましたが、やっと春の日差しが注いできました。とはいえ、朝の内は寒かったので、妻と一緒に高梁市内のスーパーに買い物とランチにでかけました。農作業が本格化する前に、リラックスしておこうというアイデアです。午後からブドウ園に行って、本剪定の作業を開始しました。仮選定の時に比べて倍くらいの時間がかかります。仮選定の時は、枝の長さを短くして本剪定の準備をすることが目的だったので、特に狙いを定める必要がなかったのでスパスパと高速に切っていくことが出来ましたが、本剪定の時には狙いを定めてうえで、所定の場所で切らなければなりません。焦らずのんびり構えて春を愉しみながら剪定作業を進めて行こうと思います。

## 2026年02月14日 AU7RS LAKSHADWEEP ISLANDS

2月10日頃からインドのチームが LAKSHADWEEP ISLANDS から QRV しています。今朝は 80m FT8 に QRV していたので、暫くコールしてなんとか QSO できました。VU7 は 160m から 10m までコンファーム済みなので、バンドニューではありませんでしたが、滅多にない DX ペディションでしか QSO できないエンティティなので、つつい呼んでしまいました。オンラインログは提供されていないようですし、QSL マネージャーの G00XO のページでも未だログを確認することはできませんでした。この DX ペディションは、DX-World によれば明日までの予定です。

## 2026年02月15日 Shrike Lite で周波数カウンタを作った

ひと月程前から Shrike Lite という RP2040 に小規模な FPGA を組み合わせたボードで遊んでいます。Verilog で回路設計をするのは初めてだったので、色々勉強したり、AI に聞いたりして、何とか周波数カウンタらしきものが出来上がりました。このままでは実用には耐えませんが、動作確認が出来たという程度のシロモノなので悪しからず。SLG47910V という FPGA には 1120 の LUT がありますが、このプロジェクトでは 203 しか使っておらず、小規模な FPGA (大きさは 5mm 角) ですが、まだまだロジックを詰め込むことができます。冒頭の画像は、パソコン側の画面 (Thonny という MicroPython の IDE でのコンソール出力) です。50MHz を 4 分周した 10MHz の信号を測定しているので、10000000Hz と表示されれば正解です。仕様は次のとおりです。

- RP2040 に MicroPython をインストールして、専用のアプリを動作させる。
- RP2040 と FPGA との間は SPI で通信し、8 ビットのコマンドを送り 32 ビットのレスポンスを得る。
- コマンドの MSB が 1 の時、測定開始を意味し、MSB が 0 の時はステータスポーリングを意味する。
- コマンドの LSB は、オンボード LED の ON/OFF を意味する。（デバッグというか動作確認用）
- レスポンスの MSB が 1 の時、ビジー（カウント中、即ちゲートが開いている状態）を意味する。
- レスポンスの MSB が 0 の時、ゲートが閉じているので、読み出した値は周波数を意味する。
- レスポンスの下位 28 ビットがカウンタの値である。
- 今の FPGA のコードでは、ゲート時間は 1 秒固定で内部クロックから生成されている。

これをベースとして、10MHz 外部基準クロックを用いれば精度が向上するでしょうし、GPS 受信機の 1PPS 信号を用いるように改造すれば、更に精度向上見込まれます。小型液晶パネルなどに結果を表示するようにすれば、パソコンなしでも利用できるようになるでしょう。今の処、周波数カウンタが欲しいわけではないので、Shrike Lite での周波数カウンタは、ここで一区切りつけたいと思います。

以下に MicroPython のソースコードと FPGA の Verilog によるソースコードを示します。

```
# Frequency Counter made with Shrike Lite
# Author: H.NAMVA 2026.02.15
import shrike
import machine
from machine import Pin, SPI
import utime
import time
shrike.flash("FreqCounter.bin")

# Reset
reset_pin = Pin(14, Pin.OUT, value=1)
```

```

reset_pin.value(0)
time.sleep(1)
reset_pin.value(1)
time.sleep(1)

# RP2040
SCK = 2
CS = 1
MOSI = 3
MISO = 0

# Chip Select pin
cs = Pin(CS, Pin.OUT, value=1)

# SPI configuration (MODE 0, MSB first)
spi = SPI(0,
          baudrate=100_000,
          polarity=0,
          phase=0,
          bits=8,
          firstbit=SPI.MSB,
          sck=Pin(SCK),
          mosi=Pin(MOSI),
          miso=Pin(MISO))

def spi_read32(cmd):
    rx = bytearray(4)
    rx = spi.read(4,cmd)
    value = int.from_bytes(rx, 'big')
    # print(f"Received {rx[0]:02x}, {rx[1]:02x}, {rx[2]:02x}, {rx[3]:02x}")
    return value

while True:
    for command in [0x81, 0x00, 0x00]:
        cs.value(0)      # Select FPGA
        resp = spi_read32(command)
        cs.value(1)      # unSelect FPGA
        if resp >= 0x80000000 :
            print(f"command: 0x{command:02X}, response: busy")
        else :
            print(f"command: 0x{command:02X}, response: {resp}Hz")
            time.sleep(1.5)

```

```

(* top *) module FreqCounter (
    (* iopad_external_pin, clkbuf_inhibit *) input clk, // System Clock (50MHz)
    (* iopad_external_pin *) output clk_en,
    (* iopad_external_pin *) input rst_n, // System Reset (Active
Low)

    // Physical SPI Pins (Connect these to FPGA I/O)
    (* iopad_external_pin *) input spi_ss_n,
    (* iopad_external_pin *) input spi_sck,
    (* iopad_external_pin *) input spi_mosi,
    (* iopad_external_pin *) output spi_miso,
    (* iopad_external_pin *) output spi_miso_en,

    (* iopad_external_pin *) output reg debug1,
    (* iopad_external_pin *) output debug1_en,
    (* iopad_external_pin *) output reg debug2,
    (* iopad_external_pin *) output debug2_en,

    // frequency counter input signal
    (* iopad_external_pin *) input freq_in,

    // Physical LED Pins
    (* iopad_external_pin *) output reg led,
    (* iopad_external_pin *) output led_en

);

assign debug1_en = 1'b1;
assign debug2_en = 1'b1;
/*
always @(posedge clk) begin
    if (spi_sck)
        debug1 <= 1'b1;
    else
        debug1 <= 1'b0;
end

always @(posedge clk) begin
    if (spi_miso)
        debug2 <= 1'b1;
    else
        debug2 <= 1'b0;
end
end

```

```

*/
    wire [27:0] freq28data;

    assign led_en = 1'b1;
    assign clk_en = 1'b1;

    wire [31:0] rx_data_wire;
    wire    rx_valid_pulse;
    reg  [31:0] tx_data_reg;

    wire tx_hold_pulse;
    wire data_ready;

    // counter logic
    always @(posedge clk) begin
        if (!rst_n) begin
            tx_data_reg <= 32'h0000_0000;
        end else if (tx_hold_pulse) begin
            tx_data_reg[27:0] <= freq28data;
            tx_data_reg[28] <= 1'b0;
                tx_data_reg[29] <= 1'b0;
                tx_data_reg[30] <= 1'b0;
                tx_data_reg[31] <= data_ready;
        end
    end

    // LED Logic
    always @(posedge clk) begin
        if (!rst_n) begin
            led <= 1'b0;
        end else if (rx_valid_pulse) begin
            if (rx_data_wire[24] == 1'b1) // if LSB of 1st byte is one
                led <= 1'b1;
            else
                led <= 1'b0;
            if (rx_data_wire[31] == 1'b1) // if MSB of 1st byte is one
                trigger <= 1'b1;
            else
                trigger <= 1'b0;
        end
    end

    //divider to generate 10MHz
    clock_divider_5 divid5(

```

```

        .clk(clk),
        .reset_n(rst_n),
        .clk_out(clk10MHz)
    );
    reg clk10MHz;

// assign debug1 = clk10MHz;
// assign debug2 = clk10MHz;
always @(posedge clk) begin
    debug1 <= clk10MHz;
    debug2 <= clk10MHz;
end

//1sec pulse generator
pulse_1s_gen p1s_gen (
    .clk(clk),
    .rst_n(rst_n),
    .clk10MHz(clk10MHz),
    .start(trigger),
    .p1s(pulse_1s)
);
reg        trigger;
wire      pulse_1s;

//28bit frequency counter
freq_counter_28bit freq28 (
    .clk(clk),
    .reset_n(rst_n),
    .sig_in(freq_in),
    .gate_in(pulse_1s),
    .freq_out(freq28data),
    .busy(data_ready)
);

// SPI Target
spi_target #(
    .CPOL(1'b0), // Standard Mode 0 (Idle Low)
    .CPHA(1'b0), // Standard Mode 0 (Sample Rising)
    .WIDTH(32),
    .LSB(1'b0) // MSB First (Standard)
) u_spi_target (
    // System Common
    .i_clk(clk),
    .i_rst_n(rst_n),
    .i_enable(1'b1), // Enable the module permanently

```

```

// SPI Physical Interface
.i_ss_n(spi_ss_n),
.i_sck(spi_sck),
.i_mosi(spi_mosi),
.o_miso(spi_miso),
.o_miso_oe(spi_miso_en),

// RX Interface (Data FROM MCU)
.o_rx_data(rx_data_wire),
.o_rx_data_valid(rx_valid_pulse),

// TX Interface (Data TO MCU)
.i_tx_data(tx_data_reg),
.o_tx_data_hold(tx_hold_pulse) // Not needed for simple echo
);

```

endmodule

```

// このモジュールはAI(Gemini)に生成してもらって、一部を変更した
module clock_divider_5 (
    input wire clk, // 元のクロック (100MHz など)
    input wire reset_n, // 同期リセット (Low アクティブ)
    output reg clk_out // 5分周された出力
);

```

```

// 状態定義 (3ビットあれば5状態を表現可能)
reg [2:0] state;

```

```

localparam S0 = 3'd0,
            S1 = 3'd1,
            S2 = 3'd2,
            S3 = 3'd3,
            S4 = 3'd4;

```

```

// ステートマシンの遷移論理

```

```

always @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        state <= S0;
        clk_out <= 1'b0;
    end else begin
        case (state)
            S0: begin
                state <= S1;
                clk_out <= 1'b1; // S0のタイミングでHighを出力
            end
            S1: begin

```

```

        state <= S2;
        clk_out <= 1'b1; // same as S1
    end
    S2: begin
        state <= S3;
        clk_out <= 1'b0;
    end
    S3: begin
        state <= S4;
        clk_out <= 1'b0;
    end
    S4: begin
        state <= S0;
        clk_out <= 1'b0;
    end
    default: begin
        state <= S0;
        clk_out <= 1'b0;
    end
endcase
end
end

```

```
endmodule
```

// このモジュールはAI(Gemini)に生成してもらって、一部を変更した

```

module freq_counter_28bit (
    input wire    clk,    // システムクロック
    input wire    reset_n, // 同期リセット (負論理)
    input wire    sig_in, // 測定対象の入力信号
    input wire    gate_in, // 外部ゲート入力
    output reg [27:0] freq_out, // 測定された周波数データ
    output reg    busy    // カウント中フラグ
);

```

```

// 内部レジスタ
reg [27:0] count_reg;
reg    gate_sync_0, gate_sync_1; // gate_in 同期化用
reg    sig_sync_0, sig_sync_1; // sig_in エッジ検出用

```

// 信号の同期化とエッジ検出用バッファ

```

always @(posedge clk) begin
    if (!reset_n) begin
        gate_sync_0 <= 1'b0;
        gate_sync_1 <= 1'b0;
        sig_sync_0  <= 1'b0;
    end
end

```

```

        sig_sync_1 <= 1'b0;
    end else begin
        gate_sync_0 <= gate_in;
        gate_sync_1 <= gate_sync_0;
        sig_sync_0 <= sig_in;
        sig_sync_1 <= sig_sync_0;
    end
end
end

// エッジ検出フラグ
wire gate_rising = (gate_sync_0 && !gate_sync_1);
wire gate_falling = (!gate_sync_0 && gate_sync_1);
wire sig_rising = (sig_sync_0 && !sig_sync_1);

// メイン制御ロジック
always @(posedge clk) begin
    if (!reset_n) begin
        count_reg <= 28'd0;
        freq_out <= 28'd0;
        busy <= 1'b0;
    end else begin
        if (gate_rising) begin
            // カウント開始
            count_reg <= 28'd0;
            busy <= 1'b1;
        end
        else if (busy) begin
            if (gate_falling) begin
                // カウント終了、結果を保持
                freq_out <= count_reg;
                busy <= 1'b0;
            end else if (sig_rising) begin
                // ゲート期間中の信号立ち上がりをカウント
                count_reg <= count_reg + 1'b1;
            end
        end
    end
end
end
end

endmodule

```

```

// このモジュールはAI(Gemini)に生成してもらって、一部を変更した
module pulse_1s_gen (
    input wire clk, // システムクロック
    input wire rst_n, // 同期リセット (負論理)
    input wire clk10MHz, // 外部 10MHz 信号

```

```

input wire start,    // 起動トリガ
output reg p1s      // 1秒パルス出力
);

// 同期化用レジスタ
reg [2:0] clk10m_sync;
reg [1:0] start_sync;

// カウンタ (10^7 = 24bit あれば十分)
reg [23:0] count;
reg      running;

// --- 1. 外部信号の同期化とエッジ検出 ---
always @(posedge clk) begin
    if (!rst_n) begin
        clk10m_sync <= 3'b0;
        start_sync  <= 2'b0;
    end else begin
        clk10m_sync <= {clk10m_sync[1:0], clk10MHz};
        start_sync  <= {start_sync[0], start};
    end
end

// clk10MHz の立ち上がりエッジ検出
wire clk10m_edge = (clk10m_sync[2:1] == 2'b01);
// start 信号の立ち上がりエッジ検出
wire start_edge  = (start_sync == 2'b01);

// --- 2. パルス生成ロジック ---
always @(posedge clk) begin
    if (!rst_n) begin
        count  <= 24'd0;
        running <= 1'b0;
        p1s    <= 1'b0;
    end else begin
        // 起動トリガ待機
        if (!running) begin
            if (start_edge) begin
                running <= 1'b1;
                p1s     <= 1'b1;
                count   <= 24'd0;
            end
        end
        // 実行中 : clk10MHz のエッジをカウント
        else if (clk10m_edge) begin
            if (count >= 24'd9_999_999) begin

```

```

        running <= 1'b0;
        p1s    <= 1'b0;
        count  <= 24'd0;
    end else begin
        count  <= count + 1'b1;
    end
end
end
end
end
end

```

```
endmodule
```

```

// 以下のSPIターゲットモジュールはRenesasのライブラリーを利用した
// 元は非同期リセットだったが、同期リセットに変更した
// -----
// c 2025 Renesas Electronics
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software
// and associated documentation files (the "Software"),
// to deal in the Software without restriction, including without limitation the rights to use, copy,
// modify, merge, publish, distribute, sublicense,
// and/or sell copies of the Software, and to permit persons to whom the Software is furnished to
// do so, subject to the following conditions:
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
// IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
// FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
// AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
// OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
// OR OTHER DEALINGS IN THE SOFTWARE.
// -----
// Base Module Name: spi_target
// Target Devices: SLG47910
// Tools version:
// Software: ForgeFPGA Workshop v.6.50
// Hardware: FPGAPAK Development Board Rev.2.0
// Revision:
// 06.09.2021 r001 - New design
// 09.19.2022 r002 - Code style review
// 03.25.2024 r003 - Changed output enable polarity
// 04.02.2024 r004 - Limits added in the parameters comments
// 10.08.2025 r005 - Redesign code. Updated module name
// -----

```

```

// Description :
// spi target (Serial Peripheral Interface) is a 4-wire synchronous serial communication interface.

//`timescale 1ns / 1ps

module spi_target #(
    parameter CPOL = 1'b0, // When one, clock is low in idle, otherwise clock is high (Type -
    Boolean, Default value = 1'b0, Min value = 1'b0, Max value = 1'b1)
    parameter CPHA = 1'b0, // When one, sampling occurs at falling edge, otherwise at rising
    edge of non-inverted clock (Type - Boolean, Default value = 1'b0, Min value = 1'b0, Max value =
    1'b1)
    parameter WIDTH = 32, // Determines the data width of SPI to the input and output data
    buses (Type - Decimal, Default value = 8, Min value = 8, Max value = 64)
    parameter LSB = 1'b0 // When one, data starts from LSB, otherwise data starts from MSB
    (Type - Boolean, Default value = 1'b0, Min value = 1'b0, Max value = 1'b1)
) (
// common ports
    input wire        i_clk,        // input clock signal
    input wire        i_rst_n,      // input negative reset signal
// control signal
    input wire        i_enable,     // input enable SPI target signal
// SPI interface ports
    input             i_ss_n,       // input target select signal
    input wire        i_sck,       // input spi clock signal
    input             i_mosi,       // input controller output target input signal
    output            o_miso,       // output controller input target output signal
    output            o_miso_oe,    // output miso enable output signal
//RX internal ports
    output reg [WIDTH-1:0] o_rx_data, // output data bus
    output reg        o_rx_data_valid, // output receive data valid signal
//TX internal ports
    input  [WIDTH-1:0] i_tx_data,    // input data bus
    output            o_tx_data_hold // output signal used to get tx data from i_tx_data input
);

// Signal declaration
    reg        [2:0] r_ss_n_sync, r_sck_sync;
    reg [$clog2(WIDTH-1):0] r_transmission_count;
    reg        [WIDTH-1:0] r_miso_data;
    wire        w_sck_r_edge, w_sck_f_edge, w_sck_edge, w_sck_edge_op;

// SPI input signals synchronization
    always @(posedge i_clk) begin
        if (!i_rst_n) begin
            r_ss_n_sync <= 'b111;
        end else if (i_enable) begin

```

```

    r_ss_n_sync <= {r_ss_n_sync[1:0], i_ss_n};
end else begin
    r_ss_n_sync <= 'b111;
end
end

always @(posedge i_clk) begin
    if (!i_rst_n) begin
        r_sck_sync <= 'h0;
    end else if (i_enable) begin
        r_sck_sync <= {r_sck_sync[1:0], i_sck};
    end else begin
        r_sck_sync <= 'h0;
    end
end

// Create rising and falling edge signals from spi_clk signal
assign w_sck_r_edge = ~r_sck_sync[2] & r_sck_sync[1];
assign w_sck_f_edge = r_sck_sync[2] & ~r_sck_sync[1];
assign w_sck_edge   = (CPHA^CPOL) ? w_sck_f_edge : w_sck_r_edge;
assign w_sck_edge_op = (CPHA^CPOL) ? w_sck_r_edge : w_sck_f_edge;

// Create transmission bit counter
always @(posedge i_clk) begin
    if (!i_rst_n) begin
        r_transmission_count <= 'h0;
    end else if (!i_enable || r_ss_n_sync[1]) begin
        r_transmission_count <= 'h0;
    end else if (w_sck_edge) begin
        if (r_transmission_count == WIDTH-1) begin
            r_transmission_count <= 'h0;
        end else begin
            r_transmission_count <= r_transmission_count + 1;
        end
    end
end

// Create o_rx_data bus and valid signals
always @(posedge i_clk) begin
    if (!i_rst_n) begin
        o_rx_data <= 'h0;
    end else if (w_sck_edge) begin
        if (LSB) begin
            o_rx_data <= {i_mosi, o_rx_data[WIDTH-1:1]};
        end else begin
            o_rx_data <= {o_rx_data[WIDTH-2:0], i_mosi};
        end
    end
end

```

```

    end
  end
end

always @(posedge i_clk) begin
  if (!i_rst_n) begin
    o_rx_data_valid <= 1'b0;
  end else if (r_ss_n_sync[1] || (r_transmission_count == 0 && w_sck_edge)) begin
    o_rx_data_valid <= 1'b0;
  end else if (w_sck_edge && r_transmission_count == WIDTH-1) begin
    o_rx_data_valid <= 1'b1;
  end
end

// Create o_tx_data_hold signal
assign o_tx_data_hold = (~CPHA & r_ss_n_sync[2] & ~r_ss_n_sync[1]) | (r_transmission_count
== 0 & w_sck_edge_op);

// Create o_miso and OE signals
always @(posedge i_clk) begin
  if (!i_rst_n) begin
    r_miso_data <= 'h0;
  end else if (o_tx_data_hold) begin
    r_miso_data <= i_tx_data;
  end else if (w_sck_edge_op) begin
    if (LSB) begin
      r_miso_data <= r_miso_data >> 1;
    end else begin
      r_miso_data <= r_miso_data << 1;
    end
  end
end

assign o_miso    = (LSB) ? r_miso_data[0] : r_miso_data[WIDTH-1];
assign o_miso_oe = ~r_ss_n_sync[2];

endmodule

```

## 2026年02月16日 AIで2列2段八木のNECファイルを作った

2年程前から430MHzでのEME用アンテナを自作するプロジェクトを始めていますが、未完成のままです。DK7ZBの23エレ八木をベースにして、共振周波数を432MHzに修正したNECファイルが手元にあります。4本を最適なスタック間隔で配置すれば、ゲインは6dB向上すると見込んでいますが、そのデータをテキストエディタで入力するのは手間が掛かるので手付かずの状態でした。スタック間隔をどのようにして決定すればよいのか知らなかったということも着手しなかった一因です。先日、スタック間隔はG/T比によって決定すべしというレターを読む機会があり、試してみようという気持ちになりました。しかし、手入力するのは面倒なので、「そうだ！AIにお願いしよう！」ということで、早速Geminiに相談したところ、対象となる八木アンテナのNECファイルをドラッグアンドドロップしてくれれば、2段2列のNECファイルを生成しますよ！との有難い返事もらったので、早速お願いしました。そのデータを元に、ビームパターンを4NEC2で計算しました。等価雑音温度の計算にはANTCというソフトを用いるらしいので早速ダウンロードしましたが、今一つ使い方が分かりません。もう少し時間がかかりそうです。

## 2026年02月17日 令和7年度確定申告

国税庁のHPを利用してe-Taxで確定申告しました。昨日から国税庁の確定申告等作成コーナーでほぼ作成完了していましたが、念のために一日置いて、本日あらためて見直してから提出することにしました。昨年の申告書と見比べると、社会保険料の控除額が少ないことに気付いたので、修正しました。国民健康保険税が丸ごと欠落していたのです。令和7年度は、株式の譲渡益がかなりあったため相当な額が源泉徴収されていました。例年は、そういう特別な収入は殆ど無いので、還付してもらうための原資となる源泉徴収を示すために、配当金や個人年金などを入力していましたが、今年度は省略しました。逆に、これらを入力すると所得が増えて来年度の住民税が増える可能性があるからです。源泉徴収された配当所得や雑所得を申告するか否かは任意なので、合法です。今年は、歯の治療にかなり掛かったのでその他の分も合わせて医療費控除が約36万円になり、その他の控除額と合わせて約194万円が控除されたので、約9万円の還付がある予定です。

## 2026年02月18日 ブドウの剪定作業完了

今日は風が強く曇りがちの天気でしたが、頑張ってぶどう園で剪定作業をしました。これでブドウの剪定作業は完了しました。この後、剪定した枝を拾い集めて焼却する作業などに移ります。ともあれ剪定作業が終われば一安心です。この後一気に暖かくなるという予報のようですが、ブリージングが早まっても大丈夫です。

## 2026年02月19日 TANG NANO 9K FPGA BOARD

デジタル回路と Verilog 入門という記事を読んで Verilog の勉強をしていたところ、この記事のなかで TANG NANO 9K という FPGA が取り上げられていました。開発ツールのインストールなどについても詳しく書かれています。この FPGA ボードを1年ほど前に秋月電子で購入していた事を思い出しました。ついついポチッとしてしまったので、箱の中でオネンネしていました。Verilog も少しは分かるようになったので、次のテーマとして取り組みたいと思います。何を作るかが問題です。

## 2026年02月20日 KP5/NP3VI on 10m

今朝、やっと KP5/NP3VI と 10m で QSO できました。サイクル 25 のピークから下降気味の最後のチャンスになんとかゲットしたいと思って、毎日のようにワッチを続けており、3日連続で呼び倒してやっと QSO に至りました。すでに ATNO として KP5/NP3VI は LoTW でコンファームできていますが、貴重なバンドニューをゲットできました。

## 2026年02月21日 ため池の水位

雨水を過ぎて春めいてきましたが、雨が少ないのが気になるところです。ネットニュースによると高知県の物部川では海に通じる河口の部分で川が干上がっていて、鮎の遡上に悪影響があるのではないかと心配されているようです。我が家のぶどう園に水を供給している溜池の水位が気になったので見に行きました。秋にサイホンの取水口を堤に取り上げた時に比べると 1m くらい水位が下がっていますが、何年か前に水漏れがひどかった時に比べると大丈夫な水位です。今もまだ水漏れしているようですが、水位がこの程度に保たれているということは、漏れている量と湧出している量がほぼバランスしているということでしょう。来週は、雨の日が多いらしいので、一安心です。

## 2026年02月22日 剪定枝を焼却

剪定作業が終わった後、拾い集めた枝の山が出来ていました。今日の天気予報によれば、午前中は風が弱く、剪定枝を焼却するには最適です。ブドウの枝は生なので簡単には火がつかず少し手間取りましたが、2時間程で綺麗に片付けました。明日は強風の予報ですし、明後日からは暫くぐずついた天気が続くようなので、今日のうちに焼却出来て良かったです。

## 2026年02月23日 キウイフルーツの剪定

毎年この時期に キウイフルーツの剪定をしています。剪定の仕方をは誰に教えてもらったわけでもなく我流でやっているのですが、今年はAI(Gemini)に写真を提示して、剪定の仕方を教えてもらいました。ググって見つけたやり方や素人っぽい人が作ったYouTube よりも良さそうな感じです。結果、例年 やってるのと大して変わりはありませんでした。スッキリと 風通し良くなったので、これでいいでしょう！

## 2026年02月24日 TangNano9k で L チカ初トライ

ShrikeLite で Verilog の書き方が少し理解できるようになったので、LUT や I/O ピン数がもう少し多い TangNano9 k に挑戦しています。TangNano9 k は、その名の通り LUT 約 9000 なので ShrikeLite の 8 倍強、30 本程度の IO ピン数を利用できるので、広範囲に応用できそうです。

最初は、ツールの使い方などを学習する目的で、LED をチカチカ点滅させる (L チカと呼ばれる) プロジェクトに取り組みました。Gowin のホームページから開発ツール (Education 版) をインストールした後、SIPEED の提供するテキストどおりに、Verilog のソースをコピペして、論理合成(Synthesize)し、ピン配置を指定して配置配線(Place& Route)した後、USB ケーブルでパソコンに接続して SRAM にプログラムしました。目出度く意図したとおりに 6 つの LED が流れるようにフラッシュすることが確認できました。ついでに、内蔵フラッシュにプログラムしてみました。ボードをパソコンから取り外して、USB ケーブルで USB 電源に接続すると、LED が点滅することが確認できました。

何を作ろうかと色々思案しています。まず思いついたのは、過去に製作したことのある 8 桁の周波数カウンタです。TangNano9k 単体で周波数カウンタは実現できますが、過去に製作したものの蒸し返しでは面白みがありません。これにラズパイ picoW を追加して、WiFi でパソコンと接続可能な周波数カウンタというのはどうでしょうか?! GPS 受信機も組み合わせれば測定精度を高めることができますし、NTS

(NetworkTimeServer) の機能を組み込むこともできそうです。もう一つのアイデアは、SDR の実験です。SDR となると A/D 変換器や D/A 変換器が必要になり、昨今入手可能な IC のパッケージの制約から製作が困難なことが予想されるので、何か良い手はないかと検討中です。

## 2026 年 02 月 25 日 Where is the 3Y0K team now?

24 名のオペレータからなる大規模なブーベ島への DX ペディションがもう直ぐ QRV する予定です。にも拘らず、どういう訳か DX-World では 3Y0K に関する情報は掲載しないとアナウンスしています。少しでも情報が欲しいのに・・・と感じている方も多いのではないかと思います。23 日になってやっと 3Y0K のホームページでニュースが更新されたようですが、更新間隔をもう少し短くして欲しいと思ったりもします。表題のセンテンスを Gemini(AI) に投げかけると、今どこに居て、いつ頃到着する予定なのかタイムリーに教えてくれます。AI には、英語で聞いた方が英語のニュースソースをメインに検索してくれるので、より多くのニュースソースから適格な情報を得ることができるそうです。How to get the position of Argus? と問いかけるとトラッキング Website を教えてくれますが、無償で観れるサイトの情報は 4 日前とかだったりして使い物になりません。衛星によるトラッキングできるサイトは有償だったので使っていません。いずれにしても、今月末頃には QRV 開始するようなので、その時を待つのみです。3Y

(Bouvet) は ANTO ではありませんが、コンファームできているのは 3 バンドのみなので、5 つ位はバンドニューをゲットしたいと期待しています。

## 2026 年 02 月 26 日 農大の親睦会

昨夕は、14 年前に通った農業大学校社会人就農研修の同窓生との親睦会に参加しました。集まったのは 7 名だけですが、皆さんそれぞれ農業に勤しんでいらっしゃる様子でした。場所は、備前市日生のつり幸という旅館でした。港のすぐ傍で、眺めの良い場所で海の幸盛り沢山の料理でした。大学時代の同窓会の場所を探していたところでしたので、ま

た来たいと思いました。写真は、鹿久居島に通じる日生大橋です。一度も渡ったことが無かったので、ついでに通ってみました。

## 2026年02月27日 杭の製作

山掃除をした時に、杭にできそうなヒノキやカシをキープしていました。何の目的で杭を作るのかというと、無線小屋からアンテナタワーまでの間の同軸ケーブルやローターのケーブル等を吊り下げるための支柱の用に供するためです。10年に一度位の頻度で更新しないと朽ちてしまいます。杭を地中に打ち込むために、先端をチェーンソーで尖らせました。ヒノキの場合、この後皮を剥かなければなりません。でないと、虫が入って簡単に朽ちてしまうからです。

## 2026年02月28日 フラクタルが美しい野菜

野菜作りは妻の担当ですが、とある野菜の最終収穫をすると言うので写真を撮らせて貰いました。この野菜はロマネスコと言って、カリフラワーやブロッコリーの仲間です。スライスして焼いてたべたりします。小さく切って焼き飯の具にすることもあります。変わった形ですが、癖のない味で、私は好きです。味もさることながら、典型的なフラクタル模様は、自然の造形に感動させられるばかりです。