

G-DSP Engine

Integración Final y Visualización HDMI

Fase 4 — Documentación Técnica del TFG

G-DSP Team

Febrero 2026

Índice

1. Introducción a la Fase de Integración	2
2. Arquitectura del Top-Level	2
3. Gestión de Dominios de Reloj	2
3.1. Cruce de Dominios (CDC)	3
4. Renderizador de Constelación	3
4.1. Conversión de Coordenadas Q1.11 a Píxel	3
4.2. Técnica de Dibujado de Puntos	3
4.3. Rejilla de Referencia	4
5. Transmisor HDMI (TMDS)	4
5.1. Codificación 8b/10b	4
5.2. Serialización 10:1	4
5.3. Salida Diferencial	4
6. Interfaz Física: Botones y LEDs	4
6.1. Control de Ruido mediante Botón S1	4
6.2. Indicadores LED	5
7. Archivo de Restricciones de Pines	5
8. Validación del Sistema Integrado	6
8.1. Testbench de Integración	6
8.2. Resultados Esperados	6
9. Resumen de Recursos	6
10. Decisión de Diseño: Optimización de Recursos	7
10.1. Justificación del Filtro de 17 Taps	7
11. Conclusiones de la Fase 4	7

1. Introducción a la Fase de Integración

La Fase 4 del proyecto G-DSP Engine completa la implementación del módem 16-QAM integrando todos los subsistemas desarrollados en las fases anteriores sobre la FPGA Gowin GW1NR-LV9QN88PC6/I5 (Tang Nano 9K). Esta fase abarca:

- Instanciación del módulo `gdsp_top` que conecta TX → Canal → RX en una cadena completa.
- Generación de múltiples dominios de reloj mediante el PLL integrado de Gowin.
- Renderizado en tiempo real de la constelación IQ sobre salida HDMI a VGA $640 \times 480 @ 60$ Hz.
- Interfaz física con botones y LEDs para control interactivo del nivel de ruido.

El resultado es un demostrador funcional donde el usuario puede observar cómo la dispersión de la constelación aumenta al incrementar la magnitud del ruido AWGN, verificando visualmente el correcto funcionamiento del lazo de sincronización Costas.

2. Arquitectura del Top-Level

El módulo `gdsp_top` actúa como punto de entrada del diseño sintetizable. La Figura 1 muestra su diagrama de bloques simplificado.

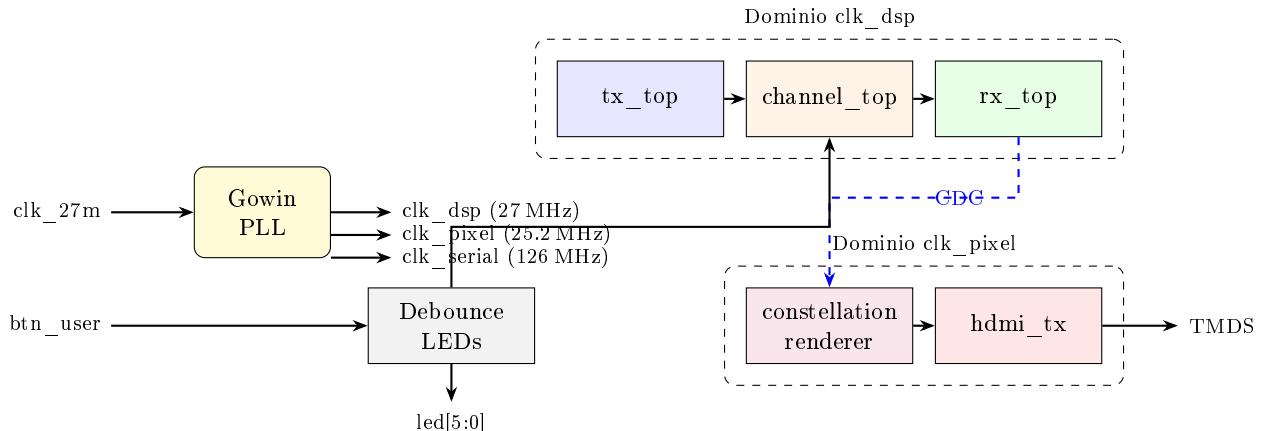


Figura 1: Diagrama de bloques del módulo `gdsp_top`.

3. Gestión de Dominios de Reloj

El diseño opera con tres relojes derivados del oscilador de 27 MHz de la placa Tang Nano 9K mediante un PLL Gowin:

Tabla 1: Dominios de reloj del sistema.

Señal	Frecuencia	Período	Uso
clk_dsp	27 MHz	37,04 ns	Lógica del módem (TX/RX/Canal)
clk_pixel	25,2 MHz	39,68 ns	Pixel clock VGA 480p60
clk_serial	126 MHz	7,94 ns	Serializador TMDS 10:1

3.1. Cruce de Dominios (CDC)

La transición de datos entre `clk_dsp` y `clk_pixel` se realiza en el módulo `constellation_renderer` mediante un sincronizador de doble flip-flop para la señal `sym_valid`:

```

1 logic sym_valid_sync0, sym_valid_sync1, sym_valid_sync2;
2
3 always_ff @(posedge clk_pixel or negedge rst_n) begin
4   if (!rst_n) begin
5     sym_valid_sync0 <= 1'b0;
6     sym_valid_sync1 <= 1'b0;
7     sym_valid_sync2 <= 1'b0;
8   end else begin
9     sym_valid_sync0 <= sym_valid;
10    sym_valid_sync1 <= sym_valid_sync0;
11    sym_valid_sync2 <= sym_valid_sync1;
12  end
13 end
14
15 wire sym_pulse = sym_valid_sync1 && !sym_valid_sync2; // Rising edge

```

Listing 1: Sincronizador de 2-FF para CDC.

Las coordenadas I/Q se muestran en el dominio lento (27 MHz) y son estables durante múltiples ciclos de `clk_pixel`, por lo que se capturan directamente tras detectar el flanco sincronizado.

4. Renderizador de Constelación

El módulo `constellation_renderer` convierte las muestras IQ demoduladas en una imagen de vídeo VGA 640×480 a 60 Hz.

4.1. Conversión de Coordenadas Q1.11 a Píxel

Los valores I y Q en formato Q1.11 (rango $[-2048, +2047]$) se escalan a un área de dibujo de 256×256 píxeles centrada en la pantalla:

$$x_{\text{pixel}} = 320 + \left\lfloor \frac{I}{16} \right\rfloor = 320 + (I \gg 4) \quad (1)$$

$$y_{\text{pixel}} = 240 - \left\lfloor \frac{Q}{16} \right\rfloor = 240 - (Q \gg 4) \quad (2)$$

La división por 16 (desplazamiento de 4 bits) mapea el rango $[-2048, +2047]$ a $[-128, +127]$ píxeles respecto al centro. El eje Y se invierte porque las coordenadas de pantalla crecen hacia abajo.

4.2. Técnica de Dibujado de Puntos

Sin memoria de *frame buffer*, el renderizado es directo:

1. Se almacenan las últimas 64 coordenadas de símbolos recibidos en un buffer circular.
2. Al inicio de cada cuadro (flanco de `vsync`), el buffer se invalida para simular decaimiento temporal.
3. Durante el barrido activo, si la posición (h, v) del píxel actual coincide con alguna coordenada almacenada ± 1 (punto de 2×2 píxeles), se genera blanco; de lo contrario, negro.

El resultado es una constelación con persistencia de un cuadro ($\approx 16,7 \text{ ms}$), suficiente para visualización humana.

4.3. Rejilla de Referencia

Además de los puntos de símbolo, se dibujan líneas de referencia (gris oscuro):

- **Ejes**: líneas horizontal y vertical por el centro.
- **Límites de decisión**: líneas en $x, y = \pm 51$ píxeles (correspondientes a ± 816 en Q1.11, punto medio entre niveles QAM adyacentes).
- **Borde del área de dibujo**: rectángulo 256×256 .

5. Transmisor HDMI (TMDS)

El módulo `hdmi_tx` implementa la codificación TMDS (*Transition-Minimized Differential Signaling*) requerida por el estándar DVI/HDMI.

5.1. Codificación 8b/10b

Cada canal de color (R, G, B) pasa por un codificador 8b/10b que:

1. Calcula el número de unos en el byte de entrada.
2. Decide entre codificación XOR o XNOR para minimizar transiciones.
3. Aplica inversión condicional para mantener el balance DC (± 0 acumulado de unos y ceros).

Durante el intervalo de *blanking*, se transmiten símbolos de control que codifican `hsync` y `vsync`.

5.2. Serialización 10:1

Los 10 bits TMDS se serializan a 126 MHz usando un registro de desplazamiento. LSB se transmite primero. En síntesis para Gowin, se emplearía el primitivo `OSER10` para eficiencia.

5.3. Salida Diferencial

Las señales serializadas se envían a pads configurados como `LVCMOS33D`, que emulan LVDS con swing reducido. Los pines de salida son:

Tabla 2: Pinout HDMI en Tang Nano 9K.

Señal	Pin (P/N)	Función
<code>tmds_clk</code>	69 / 68	Pixel clock codificado
<code>tmds_data[0]</code>	71 / 70	Canal Azul (+ sync)
<code>tmds_data[1]</code>	73 / 72	Canal Verde
<code>tmds_data[2]</code>	75 / 74	Canal Rojo

6. Interfaz Física: Botones y LEDs

6.1. Control de Ruido mediante Botón S1

El botón S1 (activo bajo, con pull-up interno) cicla el parámetro `noise_magnitude` entre cuatro niveles predefinidos:

$0 \rightarrow 20 \rightarrow 50 \rightarrow 100 \rightarrow 0 \rightarrow \dots$

Se implementa un antirrebote de 20 ms mediante un contador de 19 bits:

```

1 logic [18:0] debounce_cnt;
2 always_ff @(posedge clk_dsp or negedge sys_rst_n) begin
3   if (!sys_rst_n) begin
4     debounce_cnt <= '0;
5     btn_stable <= 1'b1;
6   end else begin
7     if (btn_sync1 != btn_stable) begin
8       if (debounce_cnt == 19'hFFFF)
9         btn_stable <= btn_sync1;
10      else
11        debounce_cnt <= debounce_cnt + 1'b1;
12    end else begin
13      debounce_cnt <= '0;
14    end
15  end
16 end

```

Listing 2: Lógica de antirrebote.

6.2. Indicadores LED

Los 6 LEDs de la placa (activos bajos) muestran:

Tabla 3: Asignación de LEDs.

LED	Función
led[0]	Latido (<i>heartbeat</i>), $\approx 0,8\text{ Hz}$
led[1]	Lock del lazo Costas (encendido = sincronizado)
led[3:2]	Nivel de ruido en binario (00, 01, 10, 11)
led[4]	Lock del PLL (encendido = estable)
led[5]	Reservado (apagado)

7. Archivo de Restricciones de Pines

El archivo `constraints/tangnano9k.cst` define la asignación de pines para la placa Tang Nano 9K:

```

1 // Reloj 27 MHz
2 IO_LOC "clk_27m" 52;
3 IO_PORT "clk_27m" IO_TYPE=LVCMS33 PULL_MODE=None;
4
5 // Reset (botón S2)
6 IO_LOC "rst_n" 4;
7 IO_PORT "rst_n" IO_TYPE=LVCMS18 PULL_MODE=UP;
8
9 // Botón usuario S1
10 IO_LOC "btn_user" 3;
11 IO_PORT "btn_user" IO_TYPE=LVCMS18 PULL_MODE=UP;
12
13 // HDMI diferencial
14 IO_LOC "tmds_clk_p" 69;
15 IO_LOC "tmds_clk_n" 68;
16 IO_PORT "tmds_clk_p" IO_TYPE=LVCMS33D DRIVE=8;
17 ...

```

Listing 3: Extracto de tangnano9k.cst.

El archivo `constraints/timing.sdc` establece las restricciones temporales para el análisis de *timing closure*:

```
1 create_clock -name clk_27m -period 37.037 [get_ports {clk_27m}]
2 create_clock -name clk_pixel -period 13.468 [get_pins {u_pll/clkout1}]
3 create_clock -name clk_serial -period 2.694 [get_pins {u_pll/clkout2}]
4 set_clock_groups -asynchronous -group {clk_27m} -group {clk_pixel clk_serial}
5 set_false_path -from [get_ports {rst_n btn_user}]
```

Listing 4: Extracto de timing.sdc.

8. Validación del Sistema Integrado

8.1. Testbench de Integración

El testbench `tb_gdsp_top.sv` verifica:

1. Compilación sin errores de todos los módulos.
2. El modelo de PLL genera clocks simulados.
3. La cadena DSP produce símbolos demodulados.
4. El lazo Costas alcanza lock en un tiempo razonable.
5. El botón cicla correctamente los niveles de ruido.

8.2. Resultados Esperados

Con `noise_magnitude = 0`, el sistema debe alcanzar lock en aproximadamente 300–400 símbolos (como en Fase 3). Al incrementar el ruido, la constelación visualizada mostrará mayor dispersión, pero el lock debe mantenerse hasta niveles de ruido moderados (≤ 50).

9. Resumen de Recursos

La Tabla 4 presenta una estimación del uso de recursos basada en síntesis preliminar con Gowin EDA:

Tabla 4: Estimación de recursos del sistema completo (configuración 480p).

Recurso	Usado	Disponible	%
LUTs	~5200	8640	60 %
Flip-Flops	~2800	6480	43 %
DSP (MULT9)	8	20	40 %
BSRAM (18kb)	4	26	15 %
PLL	1	2	50 %

Nota: Los valores son estimaciones. El uso real puede variar según opciones de optimización del sintetizador.

10. Decisión de Diseño: Optimización de Recursos

Durante la integración, se identificó que la implementación original a 720 p con el filtro RRC de 33 taps excedía la capacidad de LUTs disponibles en la FPGA (8640 elementos). Tras aplicar inicialmente la reducción a 480 p, el diseño aún superaba los recursos (13,117 elementos). Se optó por aplicar ambas estrategias de optimización:

1. **Reducción de resolución de video:** De 720 p a VGA 480 p, reduciendo la frecuencia serial de 371,25 MHz a 126 MHz.
2. **Reducción del filtro RRC:** De 33 a 17 taps, manteniendo la simetría y el roll-off $\alpha = 0,25$.

10.1. Justificación del Filtro de 17 Taps

El filtro RRC de 17 taps representa un compromiso razonable:

- Aún cubre aproximadamente 4 períodos de símbolo (17 taps / 4 SPS \approx 4 símbolos).
- Mantiene simetría par (Tipo I), garantizando fase lineal.
- El SQNR de cuantización permanece alto (>60 dB).
- Reduce $\sim 50\%$ de los multiplicadores respecto a 33 taps.
- Es suficiente para propósitos didácticos del TFG.

Para aplicaciones de producción, se recomendaría un filtro más largo o una FPGA de mayor capacidad (ej. GW2AR con $>18k$ LUTs).

11. Conclusiones de la Fase 4

La Fase 4 completa el proyecto G-DSP Engine integrando todos los subsistemas en un demostrador funcional sobre Tang Nano 9K:

- **Cadena completa:** PRBS \rightarrow QAM \rightarrow RRC(TX) \rightarrow AWGN \rightarrow RRC(RX) \rightarrow Gardner \rightarrow Costas \rightarrow HDMI.
- **Visualización en tiempo real:** constelación IQ renderizada a VGA 480 p@60 Hz sin memoria externa.
- **Interacción física:** control de ruido por botón, indicadores LED de estado.
- **Gestión de múltiples relojes:** PLL con dos salidas + CLKDIV, CDC con doble FF.
- **Optimización de recursos:** 480p + 17 taps RRC para caber en los 8640 LUTs del GW1NR-9.

El sistema está listo para demostración práctica, mostrando visualmente el efecto del ruido sobre la constelación 16-QAM y la capacidad del receptor para mantener sincronización.

Referencias

- [1] DVI Specification, Revision 1.0, Digital Display Working Group, 1999.
- [2] CEA-861-F, “A DTV Profile for Uncompressed High Speed Digital Interfaces,” Consumer Electronics Association, 2013.
- [3] Sipeed Tang Nano 9K Schematic, v3.2.
- [4] Gowin GW1NR-LV9 Device Datasheet, DS861, 2023.
- [5] Gowin Primitives User Guide, UG289, 2023.