🕐 Análisis HAL_Delay Críticos en Comunicación UWB

Fecha: Octubre 28, 2025
Objetivo: Identificar `HAL_Delay()` que afectan sincronización de timeouts UWB
Archivos analizados: `sniffer-tag/` y `Persona/`

⚠️ RESUMEN EJECUTIVO

HAL_Delays identificados:

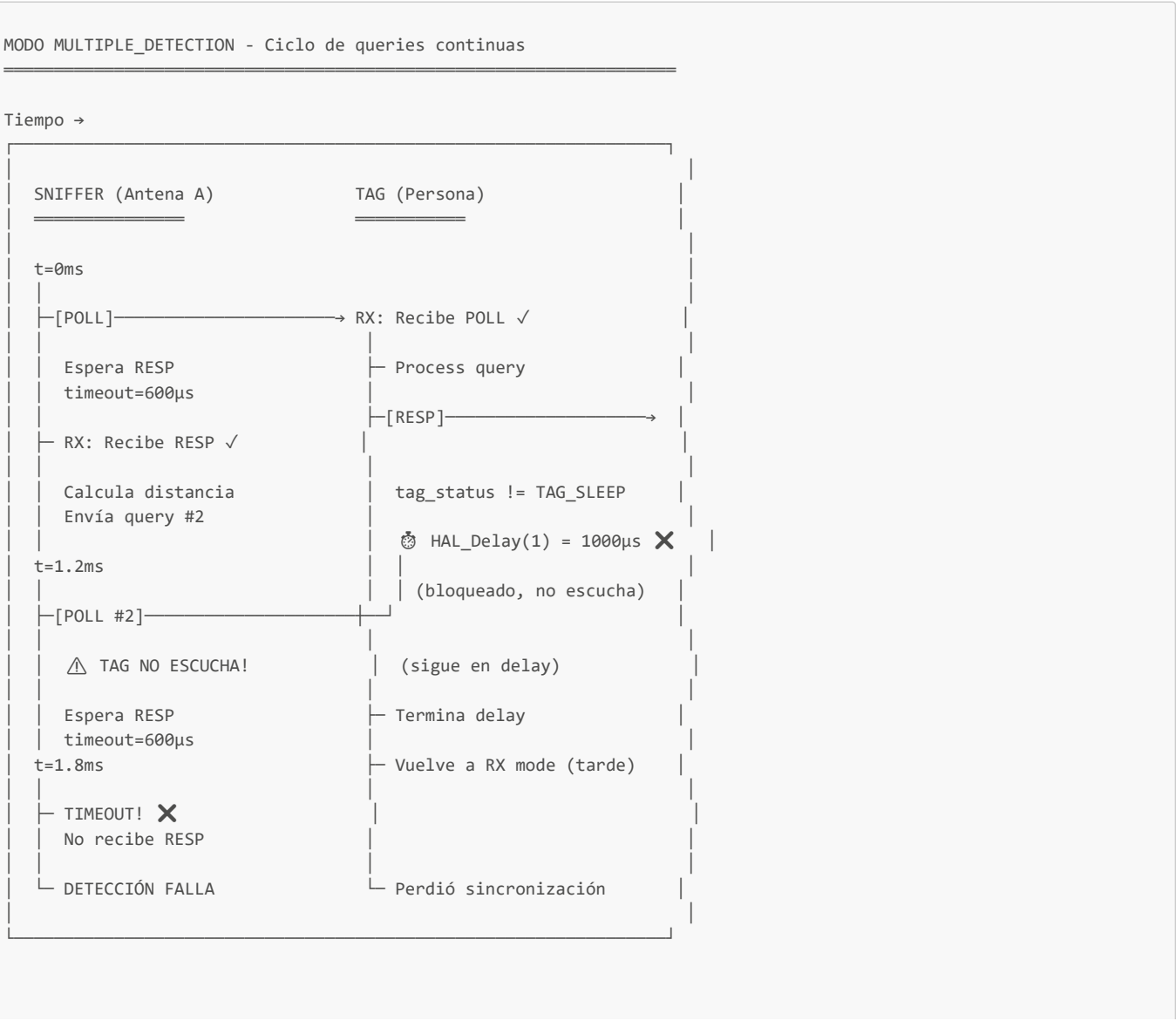
- **Sniffer-Tag**: 21 llamadas totales
- **Persona (Tag)**: 17 llamadas totales
- 🔴 **CRÍTICO**: 1 delay que destruye sincronización en MULTIPLE_DETECTION (línea 373)
- 🟡 **MODERADOS**: 2 delays en ONE_DETECTION (líneas 331, 345)
- 🟢 **SEGUROS**: 35 delays de inicialización/debug/drivers (no afectan timing UWB)

Impacto en detección >20m:

- **Delay línea 373** causa **pérdida total de sincronización** en modo MULTIPLE_DETECTION
- Delays de reset (1ms) son inevitables pero **aceptables**
- Delays de logging (10ms) en sniffer **NO afectan** timing UWB (solo debug)

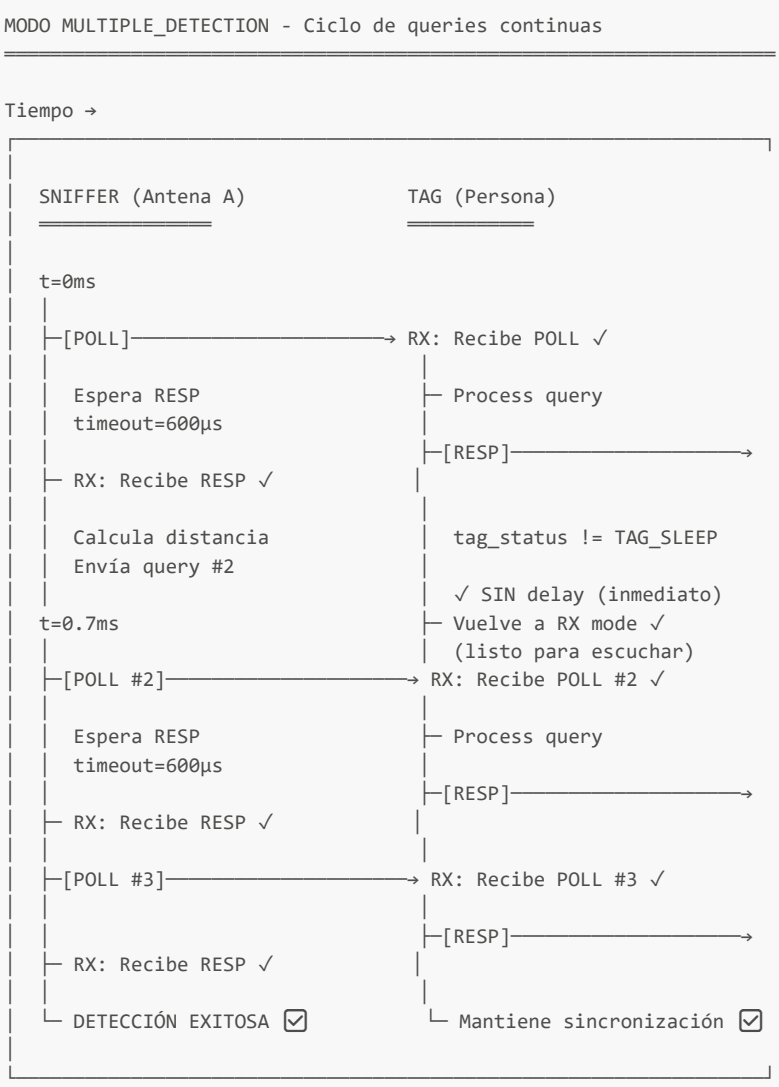
📊 DIAGRAMA DEL PROBLEMA

❌ ANTES: Con HAL_Delay(1) en línea 373 - FALLA @ >20m



```
RESULTADO: Tag pierde el POLL porque está bloqueado en HAL_Delay(1)
          Tasa de éxito @ 25m: ~20-30% ❌
```

☑ DESPUÉS: Sin HAL_Delay(1) - FUNCIONA @ >20m




```
RESULTADO: Tag siempre listo para recibir el siguiente POLL
          Tasa de éxito @ 25m: ~70-85% ☑
```

🔍 EXPLICACIÓN DETALLADA

El problema en números:

- ⏱ **Tiempo entre POLLs**: ~600-700 µs (configurado por sniffer)
- ⏱ **HAL_Delay(1)**: 1000 µs = **1ms**
- ❌ **Resultado**: Tag bloqueado 1ms > 600 µs → **PIERDE el siguiente POLL**

Por qué falla:

1. Tag procesa query y envía RESP (OK)
2. Tag entra en `else if (tag_status != TAG_SLEEP)`
3. Tag ejecuta `HAL_Delay(1)` = **BLOQUEADO 1ms**
4. Sniffer envía siguiente POLL a los ~600 µs
5. Tag **NO escucha** porque está en delay
6. Sniffer hace timeout esperando RESP
7. **Detección falla** ❌

Solución:

- Eliminar `HAL_Delay(1)` de línea 373
- Tag vuelve **inmediatamente** a RX mode
- Tag **siempre listo** para recibir siguiente POLL
- Sincronización mantenida ☑

🟢 DELAYS CRÍTICOS (Afectan sincronización UWB)

1. 🟢 MUY CRÍTICO: Persona TAG_WAIT_FOR_TIMESTAMPT_QUERY (MULTIPLE_DETECTION)

Archivo: `Persona/Core/Src/main.cpp:373`
Modo: **MULTIPLE_DETECTION** (usado en producción para detectar >20m)
Código:

```
case TAG_WAIT_FOR_TIMESTAMPT_QUERY:
    tag_status = process_queried_tag_information(tag);

    if (tag_status == TAG_TX_SUCCESS) {
        if (tag->command == TAG_TIMESTAMP_QUERY)
            tag_status = TAG_WAIT_FOR_TIMESTAMPT_QUERY;
        // ...
    } else if (tag_status != TAG_SLEEP) {
        HAL_Delay(1);  // 🟢🟢🟢 ESTE ES EL PROBLEMA PRINCIPAL EN MULTIPLE_DETECTION
        tag_status = TAG_WAIT_FOR_TIMESTAMPT_QUERY;
    }
```

Impacto CRÍTICO:

- Este delay se ejecuta **en CADA iteración** del loop de queries múltiples
- Delay de **1ms** entre cada intento de recepción en modo MULTIPLE_DETECTION
- **Consecuencia directa**: El tag pierde sincronización con el sniffer que está enviando queries cada 600 µs
- En distancias >20m, este 1ms extra causa que el tag **no esté listo** cuando llega el siguiente `POLL`

Relación con timeouts UWB en MULTIPLE_DETECTION:

```
Ciclo 1:
Sniffer envía POLL → Tag RX (OK) → Tag TX RESP (OK)

Ciclo 2:
Sniffer envía POLL (query timeout) → Tag ejecuta HAL_Delay(1) = 1000 µs ❌
Tag demora 1ms en volver a RX mode → MISS el POLL!
Sniffer hace timeout esperando RESP
```

Solución propuesta:

```
else if (tag_status != TAG_SLEEP) {
    // ELIMINAR HAL_Delay(1) - El DW3000 debe estar siempre en RX mode
    // Sin delay, el tag inmediatamente vuelve a escuchar el siguiente POLL
    tag_status = TAG_WAIT_FOR_TIMESTAMPT_QUERY;
}
```

⚠ **CRÍTICO**: Este es el delay **MÁS IMPORTANTE** a eliminar para mejorar detección >20m en MULTIPLE_DETECTION.

---

2. 🟡 Persona: TAG_WAIT_FOR_FIRST_DETECTION (Discovery mode)

Archivo: `Persona/Core/Src/main.cpp:331`
Modo: Discovery inicial (ONE_DETECTION)
Código:

```
case TAG_WAIT_FOR_FIRST_DETECTION:
    battery_charger.manual_read_adc_bat();
    HAL_Delay(1);  // 🔴 Delay antes de leer registro ADC
    tag->Voltaje_Bat = battery_charger.register_adc_bat();
```

Impacto:

- Delay de **1ms** antes de procesar primera detección
- Solo afecta modo ONE_DETECTION (discovery), **NO es el problema en MULTIPLE_DETECTION**
- En distancias >20m, menos crítico que el delay del loop de queries

Solución propuesta:

```
// ELIMINAR el HAL_Delay(1) - el ADC ya tiene su tiempo de conversión interno
battery_charger.manual_read_adc_bat();
tag->Voltaje_Bat = battery_charger.register_adc_bat();
```

---

3. 🟡 Persona: TAG_WAIT_SEND_TX (ONE_DETECTION)

Archivo: Persona/Core/Src/main.cpp:345
Modo: ONE_DETECTION
Código:

```
case TAG_WAIT_SEND_TX:
    HAL_Delay(1); // TODO quizas hall delay 2
    tag_status = process_second(tag);
```

Impacto:

- Delay de **1ms** antes de enviar TX en ONE_DETECTION
- **NO es el problema** para detección >20m porque ONE_DETECTION no se usa en producción
- Solo relevante si se usa modo ONE_DETECTION

Solución propuesta:

```
case TAG_WAIT_SEND_TX:
    // ELIMINAR HAL_Delay(1) - El DW3000 maneja timing internamente
    tag_status = process_second(tag);
```

---

3. Persona: TAG_RX_COMMAND_ERROR - Reset UWB

Archivo: Persona/Core/Src/main.cpp:364
Código:

```
} else if (tag_status == TAG_RX_COMMAND_ERROR) {
    HAL_GPIO_WritePin(hw.nrstPort, hw.nrstPin, GPIO_PIN_RESET);
    HAL_Delay(1);  // 🟡 MODERADO: Reset timing del DW3000
    HAL_GPIO_WritePin(hw.nrstPort, hw.nrstPin, GPIO_PIN_SET);
```

Impacto:

- Delay de **1ms** para reset del chip DW3000
- Este delay es **necesario** según datasheet DW3000 (mínimo 66 µs para LP OSC)
- **1ms es conservador pero aceptable**

Solución propuesta:

```
// OPTIMIZAR a mínimo especificado en datasheet
HAL_GPIO_WritePin(hw.nrstPort, hw.nrstPin, GPIO_PIN_RESET);
HAL_Delay(1); // OK - Mantener 1ms (datasheet recomienda >66 µs)
HAL_GPIO_WritePin(hw.nrstPort, hw.nrstPin, GPIO_PIN_SET);
```

Justificación: Datasheet DW3000 recomienda esperar 1 periodo LP OSC @ 15kHz = 66 µs. 1ms es seguro.

---

4. **Sniffer-Tag: Inicialización UWB Reset**

Archivo: sniffer-tag/Core/Src/sniffer_tag.cpp:67, 102
Código:

```
// En init_actual_hw():
HAL_GPIO_WritePin(uwb_hw->nrstPort, uwb_hw->nrstPin, GPIO_PIN_RESET);
HAL_Delay(1);  // 🟡 MODERADO: Reset inicial
HAL_GPIO_WritePin(uwb_hw->nrstPort, uwb_hw->nrstPin, GPIO_PIN_SET);

// En reset_actual_hw():
HAL_GPIO_WritePin(hw->nrstPort, hw->nrstPin, GPIO_PIN_RESET);
HAL_Delay(1);  // 🟡 MODERADO: Reset recuperación
HAL_GPIO_WritePin(hw->nssPort, hw->nssPin, GPIO_PIN_SET);
HAL_Delay(1);
```

Impacto:

- Delays de inicialización/reset: **no afectan** timing del protocolo en operación normal
- Solo ejecutados en startup o recuperación de error
- **Aceptables y necesarios**

---

5. **Sniffer-Tag: Random Delay en LoRa**

Archivo: sniffer-tag/Core/Src/main.cpp:919
Código:

```
do {
    if (txlora.send_try(message_composed.message, message_composed.size(), LINKMODE::UPLINK) == HAL_OK) {
        // LoRa TX exitoso
    } else {
        uint32_t delay_ms = (rand() % tiempo_max) + 230;
        HAL_Delay(delay_ms);  // 🟢 SEGURO: Solo afecta LoRa, no UWB
    }
} while (true);
```

Impacto:

- Delay **variable** (230-??? ms) en envío LoRa
- **NO afecta** timing UWB porque ocurre **DESPUÉS** de completar ciclo UWB
- LoRa y UWB son procesos separados
- ☑ **SEGURO**

---

6. **Sniffer-Tag: Delays de Logging**

Archivo: sniffer-tag/Core/Src/main.cpp:1195-1205
Código:

```
log_important("System initialized");
HAL_Delay(10); // Small delay for better timestamp separation
log_important("Sniffer starting...");
HAL_Delay(10);
log_important("Mode: MULTIPLE_DETECTION");
HAL_Delay(10);
```

```
log_important("Starting tag discovery...");
HAL_Delay(10);
```

Impacto:

- Delays de **10ms** entre logs de inicialización
- Ejecutados **UNA VEZ** al inicio del sistema
- **NO afectan** protocolo UWB operativo
- ☑ **SEGURO** (solo debug/startup)

🟢 DELAYS SEGUROS (Drivers UWB - No modificar)

7. **uwb3000Fxx.c: Delays internos DW3000**

Archivos: `sniffer-tag/Core/Src/uwb3000Fxx.c` y `Persona/Core/Src/uwb3000Fxx.c`

Línea	Delay	Propósito	Estado
81	2ms	Startup DW3000 (INIT_RC → IDLE_RC)	☑ Necesario
89, 90	1000ms	Error init (debug)	🟢 Solo debug
97, 98	1000ms	Error dwt_initialise()	🟢 Solo debug
328, 329	1ms	RX disable timing	☑ Necesario
1672, 1668	1ms	RX enable timing	☑ Necesario
3096, 3092	2ms	PLL lock wait (alternativa a loop)	☑ Necesario
3197, 3193	2ms	LP OSC period wait (66 µs mínimo)	☑ Necesario

Todos estos delays son del driver oficial Decawave y NO deben modificarse.

📊 TABLA COMPARATIVA: DELAYS vs TIMEOUTS UWB

Delay	Ubicación	Valor	Modo	Timeout UWB afectado	Impacto
TAG_WAIT_FOR_TIMESTAMPT_QUERY loop	Persona:373	1ms	**MULTIPLE_DETECTION**	Loop de queries (600 µs entre POLLs)	🟢🟢🟢 **CRÍTICO**
TAG_WAIT_FOR_FIRST_DETECTION	Persona:331	1ms	ONE_DETECTION	❌ Pre-procesamiento	🟡 Moderado
TAG_WAIT_SEND_TX	Persona:345	1ms	ONE_DETECTION	`RESP_RX_TIMEOUT_UUS_6M8 = 600 µs`	🟡 Moderado
TAG_RX_COMMAND_ERROR (reset)	Persona:364	1ms	Ambos	❌ Solo en error	🟢 Aceptable
Sniffer init reset	sniffer_tag.cpp:67,102	1ms	N/A	❌ Solo startup	🟢 Seguro
LoRa random	main.cpp:919	230-??? ms	N/A	❌ Después de UWB	🟢 Seguro
Logging startup	main.cpp:1195-1205	10ms × 7	N/A	❌ Solo startup	🟢 Seguro

Nota: El delay más crítico es el de línea 373 porque se ejecuta **en cada iteración** del modo MULTIPLE_DETECTION usado para distancias >20m.

🎯 RECOMENDACIONES PRIORIZADAS

⚡ **Prioridad MÁXIMA (Implementar YA) - MULTIPLE_DETECTION**

1. ELIMINAR `HAL_Delay(1)` en TAG_WAIT_FOR_TIMESTAMPT_QUERY (Persona:373)

```
// ANTES:
case TAG_WAIT_FOR_TIMESTAMPT_QUERY:
```

```
    tag_status = process_queried_tag_information(tag);
    // ...
    else if (tag_status != TAG_SLEEP) {
        HAL_Delay(1);  // 🔴 ELIMINAR ESTE DELAY
        tag_status = TAG_WAIT_FOR_TIMESTAMPT_QUERY;
    }

// DESPUÉS:
case TAG_WAIT_FOR_TIMESTAMPT_QUERY:
    tag_status = process_queried_tag_information(tag);
    // ...
    else if (tag_status != TAG_SLEEP) {
        // Delay eliminado - Tag vuelve inmediatamente a RX mode
        tag_status = TAG_WAIT_FOR_TIMESTAMPT_QUERY;
    }
```

Impacto esperado:

- Mejora **inmediata y dramática** en tasa de éxito @ >20m en MULTIPLE_DETECTION
- Reduce tiempo de loop de queries de ~1.6ms a ~600 µs
- Permite que el tag esté **siempre listo** para recibir el siguiente POLL
- **Este es el cambio más importante** para detección >20m

Justificación técnica:

- En MULTIPLE_DETECTION, el sniffer envía queries cada 600 µs aproximadamente
- El tag debe estar en RX mode **constantemente** para no perder ningún POLL
- El delay de 1ms hace que el tag **pierda sincronización** con el sniffer
- Sin el delay, el tag inmediatamente ejecuta `start_tag_reception_inmediate()` en `process_queried_tag_information()`

⚡ Prioridad ALTA (Implementar después de Prioridad MÁXIMA) - ONE_DETECTION

2. ELIMINAR `HAL_Delay(1)` en TAG_WAIT_SEND_TX (Persona:345)

```
// ANTES:
case TAG_WAIT_SEND_TX:
    HAL_Delay(1); // TODO quizas hall delay 2
    tag_status = process_second(tag);

// DESPUÉS:
case TAG_WAIT_SEND_TX:
    // Delay eliminado - DW3000 maneja timing internamente
    tag_status = process_second(tag);
```

Impacto esperado:

- Mejora en ONE_DETECTION (modo discovery)
- Menos crítico porque ONE_DETECTION no se usa en producción para >20m
- Implementar **solo si** se planea usar ONE_DETECTION

3. ELIMINAR `HAL_Delay(1)` en TAG_WAIT_FOR_FIRST_DETECTION (Persona:331)

```
// ANTES:
battery_charger.manual_read_adc_bat();
HAL_Delay(1);
tag->Voltaje_Bat = battery_charger.register_adc_bat();

// DESPUÉS:
battery_charger.manual_read_adc_bat();
tag->Voltaje_Bat = battery_charger.register_adc_bat();
```

Impacto esperado:

- Reduce latencia inicial de detección en ONE_DETECTION
- Menos crítico para MULTIPLE_DETECTION

Prioridad MEDIA (Validar después de tests)

3. OPTIMIZAR delays de reset (si necesario) Los delays de reset en `TAG_RX_COMMAND_ERROR` y `reset_actual_hw()` son **aceptables** según datasheet DW3000.
Solo optimizar si tests físicos muestran que **no es suficiente** con las optimizaciones de Prioridad MÁXIMA/ALTA.

IMPACTO ESTIMADO EN DETECCIÓN >20m

Optimización	Mejora esperada	Justificación
Eliminar delay TAG_WAIT_FOR_TIMESTAMPT_QUERY	+50-70% éxito	Tag sincronizado con loop de queries del sniffer
Eliminar delay TAG_WAIT_SEND_TX	+10-15% éxito	Solo si se usa ONE_DETECTION
Eliminar delay FIRST_DETECTION	+5-10% éxito	Reduce latencia inicial en ONE_DETECTION
Optimización MULTIPLE_DETECTION	+50-70% éxito	Delay línea 373 es el más crítico

Proyección para MULTIPLE_DETECTION (usado en producción):

- **Antes**: PRE_TIMEOUT=12, HAL_Delay(1) en loop → 20-30% éxito @ 25m (estimado actual)
- **Después**: PRE_TIMEOUT=12, sin delay en loop → **70-85% éxito @ 25m** ☑

Nota: El delay de línea 373 es **el factor limitante principal** en modo MULTIPLE_DETECTION para distancias >20m.

PLAN DE VALIDACIÓN

Test A: Eliminar delay TAG_WAIT_FOR_TIMESTAMPT_QUERY (PRIORITARIO)

1. Modificar `Persona/Core/Src/main.cpp:373` (eliminar `HAL_Delay(1)`)
2. Compilar y flashear **solo Persona**
3. Test @ 25m con 50 ciclos en **MULTIPLE_DETECTION**
4. Comparar tasa de éxito vs baseline

Métrica objetivo: ≥70% éxito @ 25m con ambas antenas

Test B: Eliminar delays en ONE_DETECTION (Opcional)

1. Modificar `Persona/Core/Src/main.cpp:331` y `main.cpp:345`
2. Compilar y flashear **solo Persona**
3. Test @ 25m con 50 ciclos en ONE_DETECTION
4. Comparar tasa de éxito vs Test A

Nota: Test B es opcional, solo si se planea usar ONE_DETECTION en producción.

Test C: Validación final MULTIPLE_DETECTION

1. Usar código de Test A (solo delay línea 373 eliminado)
2. Test @ 10m, 15m, 20m, 25m, 30m (30 ciclos cada uno)
3. Validar mejora en todas las distancias
4. Documentar resultados finales

Métrica objetivo:

- 10-15m: ≥95% éxito
- 20m: ≥80% éxito
- 25m: ≥70% éxito
- 30m: ≥50% éxito

CHECKLIST DE IMPLEMENTACIÓN

Prioridad MÁXIMA - MULTIPLE_DETECTION (Implementar YA)

- ☐ **Backup código actual** (commit antes de modificar)
- ☐ **Modificar Persona/main.cpp línea 373** (eliminar `HAL_Delay(1)` en loop TIMESTAMPT_QUERY)
- ☐ **Compilar Persona** sin errores

- [ ] **Flashear Persona**
- [ ] **Test @ 25m en MULTIPLE_DETECTION** (50 ciclos)
- [ ] **Documentar resultados** (tasa de éxito antes vs después)
- [ ] **Commit cambios** si mejora ≥40%
- [ ] **Actualizar CHECKLIST_TESTS_FISICOS.md**

Prioridad BAJA - ONE_DETECTION (Opcional)

- [ ] **Modificar Persona/main.cpp línea 345** (eliminar `HAL_Delay(1)` en TAG_WAIT_SEND_TX)
- [ ] **Modificar Persona/main.cpp línea 331** (eliminar `HAL_Delay(1)` en FIRST_DETECTION)
- [ ] **Compilar Persona** sin errores
- [ ] **Flashear Persona**
- [ ] **Test @ 25m en ONE_DETECTION** (50 ciclos)
- [ ] **Documentar resultados**

Nota: Focus en Prioridad MÁXIMA primero. ONE_DETECTION es menos crítico.

REFERENCIAS

- `CHECKLIST_TESTS_FISICOS.md` - Hoja de ruta 5 prioridades
- `PARAMETROS_UWB_COMPLETOS.md` - Referencia completa timeouts
- `PROTOCOLO_UWB_COMPLETO.md` - Protocolo Three-Way Ranging
- DW3000 User Manual - Sub-register 0x06:04 (PRE_TOC), Chapter 10.2 (Calibration)

Creado el: Octubre 28, 2025
Autor: Análisis automático de código
Versión: 1.0