

Perfect 👍 Let's do a **first hands-on experiment** with your **AD9545 evaluation board**:

We'll take a **25 MHz reference input** and generate a **100 MHz low-jitter output** using ACE.

---

## Step-by-Step First Experiment

### 1. Hardware Setup

- **Reference Input:**

Connect a **25 MHz clock source** (e.g., from a signal generator) to **REFA0** input SMA connector.

- Amplitude: 0.8–1.5 Vpp (single-ended) or as per datasheet.
- Use 50  $\Omega$  termination on the generator.

- **Output Clock Measurement:**

Connect **CLKOUT0** SMA to an oscilloscope or frequency counter.

- **USB Connection:**

Connect the evaluation board to your PC via USB (board will be powered from USB unless you add external supply).

---

### 2. ACE Software Setup

1. Open **ACE** on your PC.
2. The **AD9545 evaluation board** should appear in the "Attached Hardware" window. Double-click it.
3. You'll see the **Chip Diagram View** with blocks: References, PLLs, Outputs, etc.

---

### 3. Configure Reference Input

1. In the **Reference Input** block:
  - Select **REFA0**.
  - Set it to **single-ended 25 MHz**.
  - Enable it as the **primary reference**.
2. Check the **status window** → REF A should show as **valid** (green indicator).

---

### 4. Configure PLL & Loop

1. Go to **PLL0** block.
  - Set **Input** = **REFA0**.
  - Set the PLL to **lock** onto 25 MHz.
  - For fast setup, use the **PLL Wizard** (available in ACE) to calculate loop filter values automatically.
2. Configure the **PLL0 feedback divider** so that the VCO frequency is a multiple of 100 MHz (e.g., 3.2 GHz).

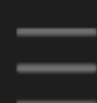
---

### 5. Configure Output

1. Go to **Output Drivers**.
  - Select **CLKOUT0**.
  - Set the divider such that from the VCO you get **100 MHz**.
  - Example: if VCO = 3.2 GHz, use divider = 32.
2. Enable the output.
  - Select format: **LVDS** or **CML** depending on hardware jumper settings.
  - Enable termination if required.

---

### 6. Apply Configuration



## 6. Apply Configuration

- Click **Download to Chip** (or **Apply All**) in ACE.
- Wait for status:
  - PLL0 should show **LOCKED**.
  - Output driver enabled.

## 7. Verify on Measurement

- On your oscilloscope/frequency counter:
  - **CLKOUT0** should show **100 MHz** stable clock.
  - Amplitude and duty cycle should match configuration.

✅ That's your **first test run**: a jitter-cleaned 100 MHz output from a 25 MHz reference.

Do you want me to also give you the **exact ACE configuration values** (PLL dividers, output divider, loop bandwidth) so you can copy-paste into ACE instead of guessing?

📄 👍 💬 ↗️ ↺ ⋮

Yes

Thought for 31s

Great — here's a **copy-paste-friendly ACE configuration** you can use to get a **25 MHz REFA** → **100 MHz CLKOUT0** working reliably on the AD9545 evaluation board. I base these settings on the AD9545 datasheet and the ADI evaluation-board notes so they'll match the device limits (VCO range, calibration steps, etc.). [Analog Devices +1](#)

## Quick summary of the approach (so the numbers make sense)


- Use the **on-board 52 MHz system crystal** (recommended in AD9545 examples) as the device system clock. This lets the AD9545 VCO and internal timing blocks operate in the expected region. [Analog Devices](#)
- Choose a **VCO (fVCO) = 2500 MHz** (inside the device examples). From that VCO you can obtain **100 MHz** by dividing by **25** ( $2500 / 25 = 100$ ). [Analog Devices](#)
- To discipline the VCO to your **25 MHz REFA**, set the DPLL feedback (N) so **N = fVCO / fREF = 2500 / 25 = 100**. That gives an integer PLL ratio with no fractional part (simple and robust). [Analog Devices](#)

## Exact ACE settings to enter

### A. System clock

1. System clock source: **On-board crystal (Xtal)** — use the on-board 52.000 MHz crystal (enable crystal/resonator path in ACE).
  - Reason: ADI examples use 52 MHz crystal; gives good oscillator/phase-noise performance. [Analog Devices](#)

### B. System clock PLL (SYSCLK)

1. Target VCO (system clock PLL output): **2,500 MHz** (this will be the device fVCO used by outputs).
2. If ACE asks for prescale/presets: follow  ACE defaults for 52 MHz crystal path.
3. After programming SYSCLK fields, run the **SYSCLK calibration** (ACE provides a button



2. Feedback (N) divider (integer part): **100** (no fractional part) → this makes  $f_{VCO} = 25 \text{ MHz} \times 100 = \mathbf{2,500 \text{ MHz}}$ . [Analog Devices](#)
3. Loop bandwidth: **50 Hz** (good default for jitter-cleaner operation; you can tighten later to reduce close-in jitter).
4. Phase buildout: **off** (unless you need alignment/phase buildout mode).
5. Reference input termination and amplitude: **~0.8 Vpp** single-ended (or follow eval board jumper settings). [Analog Devices](#)

#### D. Output driver (CLKOUT0)

1. Output source: **PLL0 (DPLL)** (the PLL you configured in C).
2. Output divider (Q or output divider register): **25** →  $2,500 \text{ MHz} / 25 = \mathbf{100 \text{ MHz}}$ .
3. Output format: choose the driver you need (e.g., **CML**, **LVDS**, or **HCSL**) — make sure the eval-board jumpers are set accordingly.
4. Drive current: **12.5 mA** is a good mid-point for CML/LVDS outputs on the eval board.
5. Enable the CLKOUT0 driver and set output termination per the eval-board manual.

[Analog Devices](#) +1

#### E. Final ACE actions (order matters)

1. In ACE: set **System clock** → choose 52 MHz crystal and **set target  $f_{VCO} = 2500 \text{ MHz}$**  (if ACE asks).
2. In ACE: configure **PLL0** (Reference = REFA0 25 MHz, N = 100, BW = 50 Hz).
3. In ACE: configure **CLKOUT0** (Source = PLL0, Output divider = 25, enable driver, set format).
4. Click **Download / Apply** (writes registers).
5. Run **SYSCLK calibrate** (ACE: “Calibrate SYSCLK” or toggle Register 0x2000 bits as ACE exposes). Wait for **SYSCLK calibration busy bit** to clear, then confirm **SYSCLK locked** and **PLL0 locked** status in ACE. [Analog Devices](#)

## Which ACE fields correspond to the numbers above

- **SYSCLK VCO target** → ACE System Clock / Crystal path pane (then calibrate). [Analog Devices](#)

- **PLL N (feedback)** → PLL0 → Feedback divider register (enter 100). [Analog Devices](#)

- **Output divider (Q)** → Output drivers → CLKOUT0 → Divider (enter 25). [Analog Devices](#)

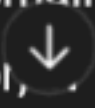
## Things to check if it doesn't lock

1. **Check REFA0 amplitude/terminations** (50  $\Omega$ , ~0.8 Vpp recommended). [Analog Devices](#)
2. **Confirm system clock is the crystal** (if you accidentally used REFA as SYSCLK the math changes).
3. **Run SYSCLK calibration** after any SYSCLK change — missing calibration is a common cause of “not locked.” [Analog Devices](#)
4. If PLL won't lock, temporarily lower loop bandwidth (wider BW like 500 Hz) to help lock, then tighten to 50 Hz once stable.
5. Use ACE status indicators: **SYSCLK locked** and **DPLL locked** bits are visible in the GUI/status window. [Analog Devices](#)

## Helpful references (useful to keep open while doing this)

- **AD9545 datasheet** (main reference, VCO ranges and examples). [Analog Devices](#)
- **AD9545 evaluation board user guide / ACE integration notes.** [wiki.analog.com](#)

If you want, I can now:

- Give the **exact ACE register writes** (a small table of register addresses + values) so you can paste into ACE's register editor, .
- Provide a **step-by-step screenshot walkthrough** of where to click in ACE (I'll give the



Good question 👍

ACE lets you **view and export the memory map (register values)** for AD9545. If you want to copy them into **Excel** for analysis or documentation, here's how you can do it:

#### ♦ Method 1: Copy-Paste from ACE GUI

1. In **ACE**, go to the **Memory Map** tab for AD9545.
2. Select the table (click in the top-left corner, or drag to highlight all).
3. Press **Ctrl + C** (Copy).
4. Open **Excel** → click in the first cell (A1) → **Ctrl + V** (Paste).
  - Excel will automatically split the **Address (Hex)** and **Data (Hex)** into separate columns.
  - If not, use **Data** → **Text to Columns** (delimiter = Tab).

#### ♦ Method 2: Export to File (Cleaner)

1. In **ACE Memory Map** window, look for **Export** or **Save As** option (usually right-click or toolbar).
2. Export as **.csv** (comma-separated) or **.txt** file.
3. Open the file directly in **Excel** → Excel will format columns automatically.
  - If text format, use **Text to Columns** (delimiter = Tab or Comma).

#### ♦ Method 3: Manual Save via Clipboard

If ACE doesn't let you export directly:

1. Copy the whole memory map (Ctrl + C).
2. Paste into **Notepad**.
3. Save the file as **ad9545\_map.csv** (change Save As Type = All Files, and put commas between columns if needed).
4. Open in Excel → it will open as a table with Address and Data columns.

✅ After this, you'll have something like:

Address (Hex)	Data (Hex)
0x0001	0x02
0x0002	0xFF
0x0003	0x10

👉 Do you want me to also give you a **ready-made Excel macro (VBA script)** that will take a plain text memory dump (Address/Data) and automatically format it into two neat columns in Excel?





Got it 👍 — in ACE's **Memory Map tab** for AD9545 there's no "export" button, but you can still get those 1000+ hex values into Excel cleanly. Here's how:

### ✅ Step 1: Copy All Values from ACE

1. Open **Memory Map** view in ACE.
2. Click anywhere inside the table.
3. Press **Ctrl + A** → this selects all rows.
4. Press **Ctrl + C** → copies the entire table (Address + Data).

### ✅ Step 2: Paste into Excel

1. Open **Excel** → start with a blank sheet.
2. Click in cell **A1**.
3. Press **Ctrl + V** → paste.

👉 Often, all data lands in **one column**. If this happens, don't worry — next step fixes it.

### ✅ Step 3: Split into Columns

1. Select the pasted column in Excel.
2. Go to **Data** → **Text to Columns**.
3. Choose **Delimited** → Next.
4. Select **Tab** (and/or **Space**) as delimiter → Finish.

Now you'll see:

Address (Hex)	Data (Hex)
0x0000	0x00
0x0001	0x1F
0x0002	0xA2

### ⚡ Alternative (Cleaner Script Method)

If ACE pastes in a messy format, you can:

- Paste into **Notepad** first.
- Save as `ad9545_dump.txt`.
- Open in Excel → Excel will auto-detect delimiters.

Or use a **VBA macro** to parse automatically.

👉 Question: Do you want me to write you a **ready-made Excel VBA macro** that will take the raw pasted memory map (even if it's messy, all in one column) and split it into proper **Address** and **Data** columns automatically?



Ahh, now I understand your pain 😊 — ACE's **AD9545 Memory Map window** is a grid control that sometimes **blocks normal copy (Ctrl+C)**. That's why you can't just select and paste 1000+ rows. Don't worry — there are a few practical workarounds so you don't need to type everything manually:

#### ◆ Option 1: ACE "Session" / XML Export

- ACE stores configuration as **session files (.xml)**.
- You can **Save Session** (from ACE top menu: *File* → *Save As*).
- That XML file has **all register addresses and values** inside.
- Then you can open it with Excel (Excel can import XML) → it will auto-create columns for **Address** and **Data**.

#### ◆ Option 2: Use ACE "Scripting Console" (Python / .NET)

- ACE has a **Scripting Console** (in the Tools menu).
- You can run a script to dump the memory map.
- Example (pseudo-Python inside ACE scripting):

```
python
```

[Copy code](#)

```
board = context.get_board("AD9545")
memmap = board.memory_map
for reg in memmap.registers:
    print(hex(reg.address), hex(reg.value))
```

- Then redirect output to a text file → import into Excel.

#### ◆ Option 3: Export Registers via "Register Debugger"

- In some ACE plug-ins (AD9545 included), the **Register Debugger** lets you "Export All Registers" to a text file.
- Check the **right-click menu** inside Memory Map, or try **Register Debugger** → **File** → **Export**.
- The exported file is CSV-like → directly openable in Excel.

#### ◆ Option 4: Screen Scraping (last resort)

If nothing else works, you can use a **free text-capture tool**:

- **NirSoft SysExporter** (Windows utility): lets you copy list-view/grid data from programs that block normal copy.
- With SysExporter you can grab the whole ACE memory map grid and dump it to a CSV file → open in Excel.

👉 [NirSoft SysExporter download ↗](#)





Perfect — that means ACE is letting you save the register map, but the format is **verbose text** (with sentences like “Register 0x0001 = 0x02 ...”), not a clean table.

👉 The good news: we can **extract just the Address and Data** from that file and put it into Excel automatically.

#### ♦ Method 1: Excel “Text to Columns” + Filters

If the text looks like:

nginx

Copy code

```
Register 0x0001 = 0x02
Register 0x0002 = 0xFF
Register 0x0003 = 0x10
```

You can do this:

1. Open the text file in **Excel** (File → Open → select your `.txt`).
2. Excel will ask how to split → choose **Delimited**.
3. Use **Space** and **=** as delimiters.
4. You’ll get columns like:

Word	Address	Equals	Data
Register	0x0001	=	0x02

5. Delete the extra columns → keep only **Address** and **Data**.

#### ♦ Method 2: Use a Simple Python Script (Fast if you’re okay with coding)

If you save the file as `dump.txt`, run this:

python

Copy code

```
output = []
with open("dump.txt") as f:
    for line in f:
        parts = line.strip().split()
        # line like: "Register 0x0001 = 0x02"
        if len(parts) >= 4 and parts[0] == "Register":
            address = parts[1]
            data = parts[3]
            output.append(f"{address},{data}")

with open("ad9545_registers.csv", "w") as f:
    f.write("Address,Data\n")
    f.write("\n".join(output))
```

You’ve hit the Free plan limit for GPT-5.

Responses will use another model until your limit resets after 11:26 AM.



Upgrade to Go



### ♦ Method 3: Excel VBA Macro (if you want everything inside Excel)

If you paste the text file contents into column A of Excel, run this macro:

vba

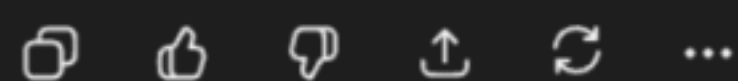
Copy code

```
Sub ParseRegisters()  
    Dim i As Long, lastRow As Long  
    lastRow = Cells(Rows.Count, 1).End(xlUp).Row  
  
    For i = 1 To lastRow  
        Dim parts() As String  
        parts = Split(Cells(i, 1).Value, " ")  
        If UBound(parts) >= 3 Then  
            Cells(i, 2).Value = parts(1) ' Address  
            Cells(i, 3).Value = parts(3) ' Data  
        End If  
    Next i  
End Sub
```

After running, column B = Address, column C = Data.

✓ This way, you don't have to type manually — it's fully automated.

Do you want me to prepare a **ready-to-use Excel file with the macro already inside** so you can just paste your text dump and click one button?

[Give c code](#)

Got it 👍 — if you'd prefer to use **C code** (instead of Python/VBA) to process the AD9545 ACE memory map text file, here's a small program.

It assumes the text file has lines like:

nginx

Copy code

```
Register 0x0001 = 0x02  
Register 0x0002 = 0xFF  
Register 0x0003 = 0x10
```

This C program will read that file, extract the **Address** and **Data**, and write them into a **CSV file** that you can open in Excel.

You've hit the Free plan limit for GPT-5.

Responses will use another model until your limit resets after 11:26 AM.

[Upgrade to Go](#)





Got it 👍 — if you'd prefer to use **C code** (instead of Python/VBA) to process the AD9545 ACE memory map text file, here's a small program.

It assumes the text file has lines like:

nginx

Copy code

```
Register 0x0001 = 0x02
Register 0x0002 = 0xFF
Register 0x0003 = 0x10
```

This C program will read that file, extract the **Address** and **Data**, and write them into a **CSV file** that you can open in Excel.

### ✓ C Code Example

c

Copy code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    FILE *fin, *fout;
    char line[256];
    char address[32], data[32];

    // Open input text file (from ACE export)
    fin = fopen("ad9545_dump.txt", "r");
    if (!fin) {
        perror("Error opening input file");
        return 1;
    }

    // Open output CSV file
    fout = fopen("ad9545_registers.csv", "w");
    if (!fout) {
        perror("Error opening output file");
        fclose(fin);
        return 1;
    }

    // Write CSV header
    fprintf(fout, "Address,Data\n");

    // Process each line
    while (fgets(line, sizeof(line), fin)) {
        // Example line: "Register 0x0001 = 0x02"
        if (sscanf(line, "Register %31s = %31s", address, data) == 2) {
            fprintf(fout, "%s,%s\n", address, data);
        }
    }
}
```

You've hit the Free plan limit for GPT-5.

Responses will use another model until your limit resets after 11:26 AM.



Upgrade to Go

```
perror("Error opening input file");
return 1;
}

// Open output CSV file
fout = fopen("ad9545_registers.csv", "w");
if (!fout) {
    perror("Error opening output file");
    fclose(fin);
    return 1;
}

// Write CSV header
fprintf(fout, "Address,Data\n");

// Process each line
while (fgets(line, sizeof(line), fin)) {
    // Example line: "Register 0x0001 = 0x02"
    if (sscanf(line, "Register %31s = %31s", address, data) == 2) {
        fprintf(fout, "%s,%s\n", address, data);
    }
}







fclose(fin);
fclose(fout);

printf("Conversion complete! Output saved to ad9545_registers.csv\n");
return 0;
}
```

#### ◆ How to Use

1. Save the C code above as `parse_ad9545.c`.
2. Put your ACE-exported text file in the same folder and name it `ad9545_dump.txt`.
3. Compile the program:
  - On Linux: `gcc parse_ad9545.c -o parse_ad9545`
  - On Windows (MinGW): `gcc parse_ad9545.c -o parse_ad9545.exe`
4. Run it:
  - `./parse_ad9545` (Linux/Mac)
  - `parse_ad9545.exe` (Windows)
5. It will create `ad9545_registers.csv`.
6. Open that CSV file in Excel → you'll see **Address** and **Data** neatly separated in two columns.

👉 Do you want me to also extend this C code so it can **directly read from ACE's XML session file** (instead of the text export)? That would give even cleaner data.

+ Ask anything

