# Trinity Trader Triangulator System

## Complete Technical Documentation

## System Overview

The Trinity Triangulator is a multi-chip IC10 system designed to locate trader ships in Stationeers by triangulating their position using satellite dish measurements. The system uses a combination of signal scanning, angle correction lookup tables, and 3D triangulation math distributed across multiple IC10 chips.

**The Complete System Architecture:**

```
┌─────────────────────────────────────────────────────────────────────────┐
│                              USER INTERFACE                              │
│                    TrinityOS (Display & State Machine)                   │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌─────────────────────────────────────────────────────────────────────────┐
│                           SEARCH CONTROLLER                              │
│                 TrinitySearch (Spiral Scan & Data Collection)            │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                    ┌───────────────┼───────────────┐
                    ▼               ▼               ▼
              Three-Shot Measurements from Different Dish Positions
                    │               │               │
                    └───────────────┼───────────────┘
                                    ▼
┌─────────────────────────────────────────────────────────────────────────┐
│                         ANGLE CORRECTION PIPELINE                        │
│    TrinityPolynomial (Table Lookup) → TrinityAngleSolver (Ratio→Angle)  │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌─────────────────────────────────────────────────────────────────────────┐
│                         TRIANGULATION PIPELINE                           │
│  TrinityOne → TrinityTwo → TrinityThree → TrinityFour → TrinityPolynomial2 │
│  (Spherical    (Build        (Cofactors &    (Solve &      (Inverse     │
│   to Cart.)    Matrix)        Determinant)    Convert)      Lookup)      │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
                          Final H, V Coordinates
                          Point Dish at Trader!
```

# Part 1: Control & Interface Scripts

## TrinityOS

### Plain English Description

TrinityOS is the **operating system and user interface** for the entire Trinity system. It manages the system state, displays status messages on LED consoles, and coordinates user interaction through buttons.

Think of it as the "brain" that tells the user what's happening:

- **Boot sequence**: Verifies all chips are properly installed by checking each named chip housing
- **Status display**: Shows scrolling text messages on LED consoles using character-by-character animation
- **State machine**: Tracks where the system is in its workflow (booting, scanning, found trader, etc.)
- **User input**: Responds to "Enter" and "Reset" button presses
- **Dish control**: Triggers interrogation of large satellite dish and trader approach sequences

The script uses a **message system** where different state numbers (stored in DB[511]) correspond to different scrolling text messages stored on the stack. When the state changes, it loads the appropriate message and scrolls it across the display character by character.

## State Machine Values (DB[511])

| State | Meaning |
|---|---|
| 0 | Reset/Reboot triggered |
| 1 | Booting (phase 1 - initializing) |
| 2 | Booting (phase 2 - loading) |
| 3 | Chip Error Detected |
| 4 | OS Nominal (all systems ready) |
| 5 | Idle/Main Menu |
| 6-7 | Waiting for user initialization |
| 8 | Three-shot procedure started |
| 9 | Data gathering in progress |
| 10 | Setting true trader location |
| 11 | Ready - prompt to interrogate |
| 12 | Interrogating dish |
| 13 | Call down prompt |
| 14 | Trader approaching |

## String Scrolling System

The OS uses a clever string display technique:

1. Messages are stored as 6-character chunks on the stack
2. Characters are extracted one at a time using bit shifting ( `srl` by 40 bits)
3. Each character is inserted into the display buffer using `ins` instruction
4. The buffer is shifted left by 8 bits to make room for the next character
5. This creates a smooth scrolling effect on the LED console

# TrinitySearch

## Plain English Description

TrinitySearch is the **main search controller** that orchestrates the entire trader-finding process. It performs a spiral scan pattern across the sky to find trader signals, then takes three measurements from different positions to triangulate the exact location.

**The search process works like this:**

### Step 1: Initialize

- Reset all systems and counters
- Point dish to starting position (H=0, V=0)
- Set dish sensitivity to 500 (wide search mode)
- Turn on both small and large satellite dishes

### Step 2: Wait for User

- Display "Select trader type" message
- User selects trader type via dial (1-255)
- User presses Enter to begin search

### Step 3: Spiral Scan

Slowly sweep the dish in an expanding spiral pattern:

- Horizontal angle increases by **0.9° per step**
- Vertical angle increases by **0.055° per step**
- At each position, check if signal matches target trader type
- Continue until match found or V reaches 90°

## Step 4: Signal Found - Three-Shot Measurement

When a matching signal is detected, take readings from 3 different dish positions:

| Shot | Horizontal | Vertical | Purpose |
|---|---|---|---|
| 1 | current + 1° | current | Baseline measurement |
| 2 | +20° from shot 1 | +2.5° | Wide horizontal offset |
| 3 | -10° from shot 2 | -5° | Triangulation point |

For each shot:

1. Move dish and wait for it to settle ( `DishMovementYield` )
2. Wait for signal to resolve ( `WaitForResolve` )
3. Read `WattsReachingContact` and dish angles
4. Send data to TrinityAngleSolver and TrinityPolynomial

## Step 5: Triangulate & Point

- Wait for math pipeline to complete (DB[12] trigger)
- Read calculated H and V from database
- Move both dishes to the true trader position
- Lock large dish's `BestContactFilter` to this trader's ID

## Step 6: Interrogate

- Display "Ready to interrogate" message
- User presses Enter to interrogate trader
- User can then call down the trader

# Part 2: Angle Correction Scripts

# TrinityPolynomial

## Plain English Description

This script performs **angle correction using a lookup table**. Satellite dishes in Stationeers don't report perfectly accurate angles—there's a non-linear relationship between what the dish reports and the true angle to the target. This script corrects for that error.

The correction table is stored in the chip's database at positions 128-218 (representing set angles 0° to 90°). When given a "set angle" (what the dish thinks it's pointing at), the script looks up the corresponding "true angle" (where it's actually pointing).

**Two modes are supported:**

- **Linear Interpolation** (mode < 0.5): Smoothly blends between table values for fractional angles
- **Step Mode** (mode ≥ 0.5): Returns the nearest lower table value (like Excel's MATCH function)

**Example:** If the dish reports 45.3°:

1. Look up the true angle for 45° (let's say it's 43.2°)
2. Look up the true angle for 46° (let's say it's 44.1°)
3. Interpolate: $43.2 + 0.3 \times (44.1 - 43.2) = 43.47^\circ$

The script processes three measurements sequentially, sending corrected angles to TrinityOne at positions 1, 5, and 9.

## Mathematical Equations

### Input

- $S$ = Set angle from dish (degrees), clamped to $[0, 90]$
- $\mathrm{mode}$ = Interpolation mode flag from DB[3]

## Table Lookup

$$n = \lfloor S \rfloor$$

$$\alpha = S - n$$

$$v_0 = \text{table}[n] = \text{DB}[128 + n]$$

$$v_1 = \text{table}[\min(n+1, 90)] = \text{DB}[128 + \min(n+1, 90)]$$

## Result Calculation

**Step Mode** (mode ≥ 0.5):

$$R = v_0$$

**Linear Interpolation** (mode < 0.5):

$$R = v_0 \cdot (1 - \alpha) + v_1 \cdot \alpha$$

## Output

$$R_{\text{final}} = \text{clamp}(R, 0.01, 89.99)$$

# TrinityPolynomial2 (Inverse Polynomial) {#trinitypolynomial2-inverse-polynomial }

## Plain English Description

This script does the **reverse of TrinityPolynomial**. Given a desired "true angle," it calculates what "set angle" you need to tell the dish to achieve that true angle.

Think of it like this:

- **TrinityPolynomial**: "I set the dish to 45°, what angle am I actually pointing?"
- **TrinityPolynomial2**: "I want to point at 43°, what should I set the dish to?"

This is essential for the final step—after triangulation calculates where the trader actually is (in true angles), this script figures out what commands to send to the dish to point there accurately (in set angles).

**The algorithm:**

1. Clamp the target angle to the valid table range [table[0], table[90]]
2. Search through the table to find which interval contains the target
3. Either return the interval start (step mode) or interpolate within the interval

## Mathematical Equations

### Input

- $R_{\text{target}}$ = Desired true angle from DB[0]
- $\text{mode}$ = Interpolation mode flag from DB[3]

### Table Search

Find smallest $n$ such that:

$$\text{table}[n+1] \geq R_{\text{target}}$$

Let:

$$v_0 = \text{table}[n]$$

$$v_1 = \text{table}[n+1]$$

## Inverse Interpolation

**Step Mode** (mode ≥ 0.5):

$$S = n, \quad \alpha = 0$$

**Linear Mode** (mode < 0.5):

$$\alpha = \frac{R_{\text{target}} - v_0}{v_1 - v_0}$$

$$\alpha = \text{clamp}(\alpha, 0, 1)$$

$$S = n + \alpha$$

## Output

$$S_{\text{final}} = \text{clamp}(S, 0, 90)$$

The result is sent to TrinitySearch at DB[44] (vertical setting) along with the horizontal angle at DB[43].

# TrinityAngleSolver

## Plain English Description

This script converts a **signal strength ratio** into an **off-axis angle measurement**. When a satellite dish detects a signal, the `WattsReachingContact` value indicates signal strength. By comparing measurements from different dish positions, we can calculate how far off-axis the target is.

The script uses **piecewise linear interpolation** to map this ratio to an angle between 0° and 180°:

- A ratio near **1.0** means the target is directly where the dish is pointing → **small angle (~0°)**
- A ratio near **0** means the target is far off to the side → **large angle (up to 180°)**

The script processes measurements for three positions sequentially, storing each calculated angle in TrinityOne at positions 2, 6, and 10.

# Mathematical Equations

## Input Processing

Let $r_1$ and $r_2$ be the two readings from the database:

$$\text{ratio} = \frac{r_2}{\max(r_1, 0.0000001)}$$

## Piecewise Angle Calculation

The angle $\theta$ is calculated based on which range the ratio falls into:

**Case 1:** If $\text{ratio} \geq 1$:

$$\theta = 0^\circ$$

**Case 2:** If $0.9 \leq \text{ratio} < 1$:

$$\theta = 2 + 3 \times \frac{1 - \text{ratio}}{0.1}$$

*Range: [2°, 5°]*

**Case 3:** If $0.1 \leq \text{ratio} < 0.9$:

$$\theta = 5 + 17.5 \times \frac{0.9 - \text{ratio}}{0.8}$$

*Range: [5°, 22.5°]*

**Case 4:** If $0.05 \leq \text{ratio} < 0.1$:

$$\theta = 22.5 + 22.5 \times \frac{0.1 - \text{ratio}}{0.05}$$

*Range: [22.5°, 45°]*

**Case 5:** If $0.01 \leq \text{ratio} < 0.05$:

$$\theta = 45 + 45 \times \frac{0.05 - \text{ratio}}{0.04}$$

*Range: [45°, 90°]*

**Case 6:** If $\text{ratio} < 0.01$:

$$\theta = 90 + 90 \times \frac{0.01 - \text{ratio}}{0.009}$$

*Range: [90°, 180°]*

## Final Output

$$\theta_{\text{final}} = \text{clamp}(\theta, 0, 180)$$

# Part 3: Triangulation Math Scripts

## TrinityOne

### Plain English Description

This script is the **coordinate converter**. It takes the spherical coordinates from each measurement position (azimuth, elevation, and the calculated angle from the correction pipeline) and converts them into **3D Cartesian unit direction vectors**.

Think of each measurement as a "cone" of possible directions where the trader could be. This script calculates the central axis of that cone as X, Y, and Z components in 3D space. It also applies an **offset** to account for which compass direction the dish base is facing.

The script processes all three measurements in sequence:

1. Reads azimuth (horizontal), elevation (vertical), corrected angle, and dish offset from database
2. Applies the offset and normalizes angles to 0-360°
3. Converts to Cartesian coordinates using trigonometry
4. Sends direction vector (x, y, z) plus weight value (w) to TrinityTwo

After processing all three measurements, it produces 12 values (4 per measurement).

### Mathematical Equations

#### Input Variables (per measurement)

- $\theta$ = Azimuth angle (horizontal direction, degrees)
- $\phi$ = Elevation angle (vertical tilt, degrees)
- $\alpha$ = Corrected angle from Polynomial (degrees)
- $\text{offset}$ = Dish facing direction offset (degrees)

## Angle Adjustment

Apply offset and normalize to [0°, 360°):

$$\theta' = ((\theta + \text{offset}) + 360) \mod 360$$

## Degree to Radian Conversion

$$k = 0.017453292519943295 = \frac{\pi}{180}$$

$$\theta_{\text{rad}} = \theta' \times k$$

$$\phi_{\text{rad}} = \phi \times k$$

$$\alpha_{\text{rad}} = \alpha \times k$$

## Cartesian Direction Vector Calculation

$$x = \sin(\theta_{\text{rad}}) \times \cos(\phi_{\text{rad}})$$

$$y = \cos(\phi_{\text{rad}})$$

$$z = \cos(\theta_{\text{rad}}) \times \cos(\phi_{\text{rad}})$$

$$w = \cos(\alpha_{\text{rad}})$$

## Output

For each measurement $i \in \{1, 2, 3\}$, output vector:

$$\vec{v}_i = (x_i, y_i, z_i, w_i)$$

Total output: 12 values stored in TrinityTwo's database at positions 0-11.

# TrinityTwo

## Plain English Description

This script builds a **least-squares system of equations**. Given three direction vectors from the measurements, we need to find where they all "intersect" in 3D space. Since real measurements have errors, they won't perfectly intersect at one point, so we use least-squares to find the best approximate solution.

The script constructs the **normal equations matrix** by accumulating products of the direction components across all three measurements. This creates a symmetric 3×3 matrix and a 3×1 right-hand side vector.

Think of it as: "Given three cones pointing in different directions, where is the point that best satisfies being on all three cones?"

## Mathematical Equations

### Input

Three direction vectors from TrinityOne:

$$\vec{v}_1 = (x_1, y_1, z_1, w_1)$$

$$\vec{v}_2 = (x_2, y_2, z_2, w_2)$$

$$\vec{v}_3 = (x_3, y_3, z_3, w_3)$$

### Building the Normal Matrix

Accumulate sums across all measurements $i = 1, 2, 3$:

**Matrix coefficients (symmetric 3×3):**

$$a = \sum_{i=1}^{3} x_i^2 \qquad b = \sum_{i=1}^{3} x_i y_i \qquad c = \sum_{i=1}^{3} x_i z_i$$

$$d = \sum_{i=1}^{3} y_i^2 \qquad e = \sum_{i=1}^{3} y_i z_i$$

$$f = \sum_{i=1}^{3} z_i^2$$

**Right-hand side vector:**

$$g = \sum_{i=1}^{3} w_i x_i$$

$$h = \sum_{i=1}^{3} w_i y_i$$

$$k = \sum_{i=1}^{3} w_i z_i$$

## Matrix Form

The system $\mathbf{A}\vec{p} = \vec{b}$:

$$\begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} g \\ h \\ k \end{pmatrix}$$

## Output

Nine values sent to TrinityThree: $(a, b, c, d, e, f, g, h, k)$

# TrinityThree

## Plain English Description

This script calculates the **cofactors and determinant** of the 3×3 matrix built by TrinityTwo. These values are needed to solve the system of equations using **Cramer's Rule**.

The cofactor matrix is essentially the "inverse ingredients" of the original matrix. The determinant tells us if the system has a unique solution (non-zero determinant) or if the measurements are degenerate (zero determinant, meaning all three measurements were taken from positions that don't provide enough geometric diversity).

Think of this as preparing all the mathematical pieces needed to divide by the matrix in the next step.

## Mathematical Equations

### Input Matrix

$$\mathbf{A} = \begin{pmatrix} a & b & c \\ b & d & e \\ c & e & f \end{pmatrix}$$

Right-hand side: $(g, h, k)$

### Cofactor Calculation

The cofactors of matrix $\mathbf{A}$:

$$C_{11} = df - e^2$$

$$C_{12} = ce - bf \quad \text{(note: used as } bf - ce \text{ with sign handled in solve)}$$

$$C_{13} = be - cd$$

$$C_{22} = af - c^2$$

$$C_{23} = bc - ae$$

$$C_{33} = ad - b^2$$

## Determinant

Using cofactor expansion along the first row:

$$\det(\mathbf{A}) = a \cdot C_{11} + b \cdot C_{12} + c \cdot C_{13}$$

Expanded:

$$\det(\mathbf{A}) = a(df - e^2) + b(ce - bf) + c(be - cd)$$

## Output

Ten values sent to TrinityFour:

$$(C_{11}, C_{12}, C_{13}, C_{22}, C_{23}, C_{33}, \det(\mathbf{A}), g, h, k)$$

# TrinityFour {#trinityfour }

## Plain English Description

This is the **final solver** that computes the target direction. It uses **Cramer's Rule** to solve the linear system, producing a 3D position vector. Then it normalizes this vector to unit length and converts it back to **spherical coordinates** (azimuth and elevation angles).

The output angles are "true angles" - where the trader actually is in the sky. These are then sent to TrinityPolynomial2 to convert back to "set angles" that the dish understands.

The output is:

- **Horizontal angle (H)**: The compass direction to the target, wrapped to 0-360°
- **Vertical angle (V)**: The elevation angle, clamped to 0-180°

## Mathematical Equations

### Input

From TrinityThree:

- Cofactors: $C_1, C_2, C_3, C_4, C_5, C_6$
- Determinant: $D$
- Right-hand side: $g, h, k$

### Solve Using Cramer's Rule

$$p_x = \frac{C_1 \cdot g + C_2 \cdot h + C_3 \cdot k}{D}$$

$$p_y = \frac{C_2 \cdot g + C_4 \cdot h + C_5 \cdot k}{D}$$

$$p_z = \frac{C_3 \cdot g + C_5 \cdot h + C_6 \cdot k}{D}$$

### Normalize to Unit Vector

$$\text{magnitude} = \sqrt{p_x^2 + p_y^2 + p_z^2}$$

$$\hat{x} = \frac{p_x}{\text{magnitude}}$$

$$\hat{y} = \frac{p_y}{\text{magnitude}}$$

$$\hat{z} = \frac{p_z}{\text{magnitude}}$$

## Convert to Spherical Coordinates

**Horizontal distance in XZ plane:**

$$d_h = \sqrt{\hat{x}^2 + \hat{z}^2}$$

**Azimuth (horizontal angle):**

$$H_{\text{rad}} = \text{atan2}(\hat{z}, \hat{x})$$

$$H_{\text{deg}} = H_{\text{rad}} \times \frac{180}{\pi} = H_{\text{rad}} \times 57.29577951308232$$

**Adjusted azimuth (convert from atan2 convention):**

$$H = (90^\circ - H_{\text{deg}}) \mod 360^\circ$$

**Elevation (vertical angle):**

$$V_{\text{rad}} = \text{atan2}(d_h, \hat{y})$$

$$V_{\text{deg}} = V_{\text{rad}} \times 57.29577951308232$$

$$V = \text{clamp}(V_{\text{deg}}, 0^\circ, 180^\circ)$$

## Final Output

$$\boxed{H = \text{Azimuth (compass direction)} \in [0^\circ, 360^\circ)}$$

$$\boxed{V = \text{Elevation angle} \in [0^\circ, 180^\circ]}$$

These values are sent to TrinityPolynomial2 for inverse lookup to get dish settings.

# Part 4

# Complete System Integration

# How It All Works Together

## The Complete Workflow

```
┌─────────────────────────────────────────────────────────────────────────┐
│  PHASE 1: BOOT & INITIALIZATION                                         │
│  ═══════════════════════════════                                        │
│                                                                         │
│  TrinityOS boots up and checks all chip housings:                       │
│    → Verifies TrinitySearch, TrinityAngleSolver, TrinityPolynomial, etc.│
│    → Displays "BOOTING..." then "SYSTEMS NOMINAL" on LED console        │
│    → Enters idle state, waiting for user                                │
│                                                                         │
│  User selects trader type on dial (1=Food, 2=Medical, etc.)             │
│  User presses ENTER button                                              │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
┌─────────────────────────────────────────────────────────────────────────┐
│  PHASE 2: SPIRAL SEARCH                                                 │
│  ══════════════════════                                                 │
│                                                                         │
│  TrinitySearch takes control:                                           │
│    → Sets dish to wide-beam mode (Setting = 500)                        │
│    → Begins spiral scan: H += 0.9°, V += 0.055° each tick               │
│    → Reads TraderData at each position                                  │
│    → Compares signal type to user's selection                           │
│                                                                         │
│  Display shows: "SCANNING..."                                           │
│                                                                         │
│  When matching trader found → Proceed to Phase 3                        │
│  If V reaches 90° with no match → Reset and try again                   │
└─────────────────────────────────────────────────────────────────────────┘
                                    │
                                    ▼
```

```
┌───────────────────────────────────────────────────────────────────────────────┐
│  PHASE 3: THREE-SHOT DATA COLLECTION                                          │
│  ═══════════════════════════════════                                          │
│                                                                               │
│  TrinitySearch switches to precision mode (Setting = 7500)                    │
│                                                                               │
│  SHOT 1: Move dish to (H+1°, V)                                               │
│    → Wait for dish to settle                                                  │
│    → Read WattsReachingContact → Send to TrinityAngleSolver DB[1]             │
│    → Read dish V angle → Send to TrinityPolynomial DB[0]                      │
│    → Read dish H angle → Send to TrinityPolynomial DB[5]                      │
│    → Trigger both chips (DB[12] = 1)                                          │
│                                                                               │
│  SHOT 2: Move dish to (H+21°, V+2.5°)                                         │
│    → Same data collection process                                             │
│                                                                               │
│  SHOT 3: Move dish to (H+11°, V-2.5°)                                         │
│    → Same data collection process                                             │
│    → This triggers the full triangulation pipeline                            │
└───────────────────────────────────────────────────────────────────────────────┘
                                        │
                                        ▼
┌───────────────────────────────────────────────────────────────────────────────┐
│  PHASE 4: ANGLE CORRECTION (runs 3 times, once per shot)                      │
│  ═══════════════════════════════════════════════════════                      │
│                                                                               │
│  For each shot:                                                               │
│                                                                               │
│  TrinityPolynomial:                                                           │
│    → Reads set angle S from DB[0]                                             │
│    → Looks up in table: DB[128+floor(S)]                                      │
│    → Interpolates to get true angle R                                         │
│    → Sends R to TrinityOne at position 1, 5, or 9                             │
│    → Sends H to TrinityOne at position 0, 4, or 8                             │
│                                                                               │
│  TrinityAngleSolver (in parallel):                                            │
│    → Reads signal strengths from DB[0], DB[1]                                 │
│    → Calculates ratio and maps to off-axis angle θ                            │
│    → Sends θ to TrinityOne at position 2, 6, or 10                            │
│                                                                               │
│  After 3rd shot: TrinityPolynomial triggers TrinityOne (DB[12] = 1)           │
└───────────────────────────────────────────────────────────────────────────────┘
```

```
                                   │
                                   ▼
┌─────────────────────────────────────────────────────────────────────────┐
│  PHASE 5: TRIANGULATION PIPELINE                                        │
│  ═══════════════════════════════                                        │
│                                                                         │
│  TrinityOne (triggered):                                                │
│    → Reads 12 values: (H, V_true, θ) × 3 shots + offsets                │
│    → Converts each to Cartesian: (x, y, z, w)                           │
│    → Sends 12 values to TrinityTwo                                      │
│    → Triggers TrinityTwo                                                │
│                                                                         │
│  TrinityTwo (triggered):                                                │
│    → Accumulates: Σx², Σxy, Σxz, Σy², Σyz, Σz², Σwx, Σwy, Σwz           │
│    → Sends 9 matrix values to TrinityThree                              │
│    → Triggers TrinityThree                                              │
│                                                                         │
│  TrinityThree (triggered):                                              │
│    → Calculates 6 cofactors and determinant                             │
│    → Sends 10 values to TrinityFour                                     │
│    → Triggers TrinityFour                                               │
│                                                                         │
│  TrinityFour (triggered):                                               │
│    → Solves system using Cramer's Rule → (px, py, pz)                   │
│    → Normalizes to unit vector                                          │
│    → Converts to spherical: H_true, V_true                              │
│    → Sends V_true to TrinityPolynomial2 DB[0]                           │
│    → Sends H_true to TrinityPolynomial2 DB[43]                          │
│    → Triggers TrinityPolynomial2                                        │
└─────────────────────────────────────────────────────────────────────────┘
                                   │
                                   ▼
```

```
┌─────────────────────────────────────────────────────────────────────────────┐
│  PHASE 6: INVERSE LOOKUP & FINAL POSITIONING                                │
│  ═══════════════════════════════════════════                                │
│                                                                             │
│  TrinityPolynomial2 (triggered):                                            │
│    → Reads V_true from DB[0]                                                │
│    → Searches table backwards to find S that produces V_true                │
│    → Interpolates to get exact V_set                                        │
│    → Sends V_set to TrinitySearch DB[44]                                    │
│    → Sends H (passed through) to TrinitySearch DB[43]                       │
│    → Triggers TrinitySearch (DB[12] = 1)                                    │
│                                                                             │
│  TrinitySearch receives final coordinates:                                  │
│    → Reads H_set from DB[43], V_set from DB[44]                             │
│    → Moves BOTH dishes to (H_set, V_set)                                    │
│    → Locks large dish filter to this trader's SignalID                      │
│    → Updates display: "TRADER FOUND"                                        │
└─────────────────────────────────────────────────────────────────────────────┘
                                        │
                                        ▼
┌─────────────────────────────────────────────────────────────────────────────┐
│  PHASE 7: INTERROGATION & APPROACH                                          │
│  ═════════════════════════════════                                          │
│                                                                             │
│  TrinityOS displays: "PRESS ENTER TO INTERROGATE"                           │
│                                                                             │
│  User presses ENTER:                                                        │
│    → Large dish Setting = 50000 (tight beam)                                │
│    → Large dish Activate = 1 (begin interrogation)                          │
│    → Wait for InterrogationProgress = 1                                     │
│                                                                             │
│  TrinityOS displays: "PRESS ENTER TO CALL DOWN"                             │
│                                                                             │
│  User presses ENTER:                                                        │
│    → Small dish Activate = 1 (signal approach)                              │
│    → Display: "TRADER APPROACHING"                                          │
│    → Trader ship descends to base!                                          │
│                                                                             │
│  User can press RESET at any time to start over                             │
└─────────────────────────────────────────────────────────────────────────────┘
```

# Data Flow Summary

```
                    ┌─────────────┐
                    │  Satellite  │
                    │    Dish     │
                    └──────┬──────┘
                           │
         ┌─────────────────┼─────────────────┐
         │                 │                 │
         ▼                 ▼                 ▼
      Shot 1            Shot 2            Shot 3
    (H+1, V)       (H+21, V+2.5)     (H+11, V-2.5)
         │                 │                 │
         └─────────────────┼─────────────────┘
                           │
        ┌──────────────────┴──────────────────┐
        │                                     │
        ▼                                     ▼
┌──────────────────┐                 ┌──────────────────┐
│ TrinityPolynomial│                 │TrinityAngleSolver│
│  (V → V_true)    │                 │ (Watts → θ)      │
└───────┬──────────┘                 └────────┬─────────┘
        │                                     │
        │      ┌──────────────────────────────┘
        │      │
        ▼      ▼
    ┌──────────────────┐
    │   TrinityOne     │
    │ (Spherical→Cart) │
    └────────┬─────────┘
             │  (x,y,z,w) × 3
             ▼
    ┌──────────────────┐
    │   TrinityTwo     │
    │ (Build Matrix)   │
    └────────┬─────────┘
             │  A, b
             ▼
    ┌──────────────────┐
    │  TrinityThree    │
    │ (Cofactors/Det)  │
    └────────┬─────────┘
```

```
        │  C, det
        ▼
┌─────────────────┐
│   TrinityFour   │
│  (Solve→H,V)    │
└────────┬────────┘
         │  H_true, V_true
         ▼
┌─────────────────┐
│TrinityPolynomial2│
│ (V_true → V_set)│
└────────┬────────┘
         │  H_set, V_set
         ▼
┌─────────────────┐
│  TrinitySearch  │
│  (Point Dish)   │
└────────┬────────┘
         │
         ▼
   🛸  TRADER FOUND!  🛸
```

# Chip Communication Protocol

All chips use database slot 12 as a **trigger flag** and slot 13 as a **counter** for the three-shot sequence:

| Chip | Reads DB[12] | Writes DB[12] | Counter DB[13] |
|---|---|---|---|
| TrinitySearch | Waits for completion | Triggers pipeline | - |
| TrinityPolynomial | 1 = process | 0 = done | Tracks shot 1,2,3 |
| TrinityAngleSolver | 1 = process | 0 = done | Tracks shot 1,2,3 |
| TrinityOne | 1 = process | 1 → TrinityTwo | - |
| TrinityTwo | 1 = process | 1 → TrinityThree | - |
| TrinityThree | 1 = process | 1 → TrinityFour | - |
| TrinityFour | 1 = process | 1 → Polynomial2 | - |
| TrinityPolynomial2 | 1 = process | 1 → Search | - |

**Special trigger values:**

- `DB[12] = 3` sent to AngleSolver or Polynomial = Reset counters

# Hardware Requirements

| Component | Name Hash | Purpose |
|---|---|---|
| Small Satellite Dish | `StructureSatelliteDish` | Scanning & signaling |
| Large Satellite Dish | `StructureLargeSatelliteDish` | Interrogation |
| Logic Button (Enter) | Name: "Enter" | User input |
| Logic Button (Reset) | Name: "Reset" | Abort/restart |
| Logic Dial | Any | Trader type selection |
| LED Console 1×3 | Name: "TrinityOS" | Status display |
| 8× Circuit Housing | Named per script | Chip holders |

**Chip Housing Names:**

- TrinityOS
- TrinitySearch
- TrinityAngleSolver
- TrinityPolynomial
- TrinityPolynomial2
- TrinityOne
- TrinityTwo
- TrinityThree
- TrinityFour

*Document generated for the Stationeers IC10 Trinity Trader Triangulator System*