MEMORY SHARE Command User Manual

Overview

The `MEMORY SHARE` command provides a high-speed, continuous one-way data link between two PicoMite boards using PIO and DMA hardware. A **host** board streams the contents of a memory buffer to a **client** board over 8 data lines and a clock line. The transfer runs entirely in hardware after startup ? no CPU cycles are consumed during operation.

The host repeatedly transmits the contents of its shared memory buffer in a continuous loop. The client's buffer is automatically updated with the latest data from the host. This allows the client to read the host's memory at any time and always see a recent copy.

Syntax

Start Host (Transmitter)

```
MEMORY SHARE HOST pio, sm, data_pin, clock_pin, address, count [, clock_div]
```

Start Client (Receiver)

```
MEMORY SHARE CLIENT pio, sm, data_pin, clock_pin, address, count
```

Stop

```
MEMORY SHARE STOP
```

Parameters

Parameter	Description
`pio`	PIO block to use: 0, 1, or 2 (RP2350 only). The selected PIO block must be available.
`sm`	State machine number within the PIO block (0 to 3).
`data_pin`	First of 8 consecutive GPIO pins used for the data bus. The next 7 pins are used automatically. Can use GP notation (e.g., `GP0`).
`clock_pin`	GPIO pin used for the clock signal. Must not overlap the 8 data pins.
`address`	Start address of the shared memory buffer. Use `PEEK(VARADDR variable())` to get the address of an array.
`count`	Number of bytes to transfer per cycle. Must be greater than 0 and a multiple of 4.
`clock_div`	(Host only, optional) PIO clock divider. Default is 10. Minimum is 3. Higher values reduce transfer speed but improve signal integrity over longer wires.

Wiring

The host and client must be connected with 9 wires plus a ground:

Host Pin	Client Pin	Function
GP*n*	GP*n*	Data bit 0
GP*n+1*	GP*n+1*	Data bit 1
GP*n+2*	GP*n+2*	Data bit 2
GP*n+3*	GP*n+3*	Data bit 3
GP*n+4*	GP*n+4*	Data bit 4
GP*n+5*	GP*n+5*	Data bit 5
GP*n+6*	GP*n+6*	Data bit 6
GP*n+7*	GP*n+7*	Data bit 7
GP*clk*	GP*clk*	Clock
GND	GND	Ground

Notes:
- The data pin numbers on the host do not need to match the client ? they just need to be physically wired together.
- The clock pin can be any GPIO that does not overlap the 8 data pins.
- Keep wires short (under ~20 cm) for reliable high-speed operation. For longer wires, increase the clock divider.

Handshake and Synchronisation

The host and client perform an automatic handshake at startup and can be started in any order:

1. **Host** drives a sync pattern (`0101`) on data pins 0?3 and pulls the clock LOW. It then waits for the client to respond.
2. **Client** checks that the host clock is not already toggling. If it is, an error is raised (the host was already streaming and the client cannot synchronise). Otherwise, the client drives `1010` on data pins 4?7 and waits to see the host's `0101` on pins 0?3.
3. Once both sides see the other's pattern, they hold for 1 ms then proceed. The client switches its pins to PIO receive mode immediately. The host waits an additional 5 ms to ensure the client is ready, then begins streaming.

This means:
- **Initial startup (both sides stopped): either side can be started first.** The first one started will block at the console until the other side starts.
- Press **Ctrl-C** to cancel a waiting handshake.
- If the host is already streaming when the client starts, the client will error with `"Host already running - cannot synchronise"`. Stop the host first, then restart both.

Transfer Rate

The host PIO program uses 2 instructions per byte: one to output data (clock LOW) and one to hold (clock HIGH). The transfer rate is:

$$\text{Byte rate} = \frac{f_{sys}}{2 \times \text{clock\_div}} \quad \text{bytes/sec}$$

Clock Divider	Byte Rate (at 125 MHz)	Notes
10 (default)	6.25 MB/s	Conservative, reliable with breadboard wiring
4	15.6 MB/s	Recommended maximum for short direct connections
3 (minimum)	20.8 MB/s	Fastest allowed setting ? requires very short wires

The minimum clock divider is **3**, enforced by the firmware. Below this value the client cannot reliably track clock edges through the GPIO input synchroniser (2 system clocks latency) plus the 3-instruction receive loop. The client PIO runs at full system clock speed (divider = 1).

Stopping

```
MEMORY SHARE STOP
```

This stops the PIO state machine, aborts both DMA channels, resets all 9 GPIO pins to their default state (SIO input, no pulls), releases pin reservations, and removes the PIO program. It is safe to call `MEMORY SHARE STOP` even when no share is active.

Running `MEMORY SHARE HOST` or `MEMORY SHARE CLIENT` a second time automatically stops any previous share before starting the new one.

Error Messages

Error	Cause
`Count must be > 0 and multiple of 4`	The `count` parameter is zero, negative, or not divisible by 4.
`Clock divider must be >= 3`	The `clock_div` parameter is less than 3.
`Need 8 consecutive GPIOs from data pin`	The 8 pins starting at `data_pin` would exceed GPIO 29.
`Clock pin overlaps data pins`	The clock GPIO falls within the range of the 8 data GPIOs.
`No PIO instruction space available`	The selected PIO block has no room for the share program. Other PIO programs may need to be stopped first.
`Host already running - cannot synchronise`	(Client only) The host clock is already toggling. Stop the host, then restart both sides.

Example

Host Program (RP2040)

```
' Host - transmits 100 integers to the client
Option Base 0
On Error Skip
```

```
Memory Share Stop

Dim shared%(99)
Dim addr% = Peek(VarAddr shared%())

' Fill with test data
Dim i%, counter%
For i% = 0 To 99
  shared%(i%) = i%
Next

' Start host: PIO 1, SM 0, data GP0, clock GP8, 800 bytes, divider 10
Memory Share Host 1, 0, GP0, GP8, addr%, 800, 10

Print "Host running. Press any key to stop."

' Continuously update shared data
counter% = 0
Do
  For i% = 0 To 99
    shared%(i%) = counter% + i%
  Next
  counter% = counter% + 100
  Pause 500
Loop Until Inkey$ <> ""

Memory Share Stop
Print "Stopped."
```

Client Program (RP2350)

```
' Client - receives 100 integers from the host
Option Base 0
On Error Skip
Memory Share Stop

Dim shared%(99)
Dim addr% = Peek(VarAddr shared%())

' Clear buffer
Dim i%
For i% = 0 To 99
  shared%(i%) = -1
Next

' Start client: PIO 1, SM 0, data GP0, clock GP15, 800 bytes
Memory Share Client 1, 0, GP0, GP15, addr%, 800

Print "Client running. Press any key to stop."

' Display received data
Do
  Print "first=" Hex$(shared%(0));
  Print "  [50]=" Hex$(shared%(50));
  Print "  last=" Hex$(shared%(99))
  Pause 500
Loop Until Inkey$ <> ""

Memory Share Stop
Print "Stopped."
```

Technical Details

Hardware Resources Used

Each active share uses:
- 1 PIO state machine (2 instruction slots for host, 3 for client)

- 2 DMA channels (channels 10 and 11): one for data transfer, one for address reset and retrigger
- 9 GPIO pins (8 data + 1 clock)

How It Works

Host: DMA continuously transfers 32-bit words from the shared memory buffer to the PIO TX FIFO. The PIO state machine shifts each word out 8 bits at a time on the data pins, toggling the clock via sideset. When the data channel completes one full buffer, it chains to the control channel, which resets the read address and retriggers the data channel ? creating an infinite loop.

Client: The PIO state machine waits for the host's clock edges and shifts in 8 bits at a time. After 4 bytes, autopush moves the assembled 32-bit word to the RX FIFO. DMA drains the RX FIFO into the shared memory buffer. The same chain-and-retrigger mechanism provides continuous operation.

Data Coherency

The transfer is **not synchronised** with the host's BASIC program. This means:
- The client may see a mix of old and new data if the host is updating the buffer while transfer is in progress.
- For applications requiring coherent snapshots, use a sequence counter or double-buffering scheme in your BASIC program. For example, write a counter as the first and last element of the buffer. The client can verify both match to confirm a complete, consistent block was received.

Re-running

Both sides can be stopped and restarted without power-cycling. The startup sequence fully cleans up any stale PIO programs, DMA state, and GPIO configuration from a previous run.

For re-running a host/client pair, use this order:
1. Start (or restart) the **host** first to re-initialise the sync state.
2. Start (or restart) the **client** second.

If the client is started first while the previous host instance is still streaming in the background, the client cannot resynchronise and will fail with `Host already running - cannot synchronise`.

Use `MEMORY SHARE STOP` on each side when needed, or simply run a new local `MEMORY SHARE HOST`/`CLIENT` command ? the previous local share is stopped automatically before the new one starts.

Pin Constraints

- The 8 data pins must be **consecutive GPIOs** (e.g., GP0?GP7, GP10?GP17).
- The clock pin must be **outside** the data pin range.
- On RP2350 with the A9 errata, the firmware automatically enables pad input on all pins used by the client.
- The `WAIT GPIO` PIO instruction limits the clock pin to GPIO 0?31.