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 a parallel data bus and a clock line. The transfer runs entirely in hardware after startup -- no CPU cycles are consumed during operation.

The data bus can be either 8 bits wide (default, using 8 data pins) or 4 bits wide (using 4 data pins). The 4-bit mode uses fewer GPIO pins at the cost of halving the transfer rate.

A single PicoMite can run both a host and a client simultaneously, enabling full-duplex (bidirectional) data exchange between two boards. The host and client use separate DMA channels, PIO state machines, and GPIO pins, so they operate completely independently. This allows two boards to share data in both directions at the same time.

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.

Syntax

Start Host (Transmitter)

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

Start Client (Receiver)

```
MEMORY SHARE CLIENT pio, sm, data_pin, clock_pin, address, count [, bus_width]
```

Stop

```
MEMORY SHARE STOP          ' stops both host and client
MEMORY SHARE STOP HOST     ' stops host only
MEMORY SHARE STOP CLIENT   ' stops client only
```

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 the consecutive GPIO pins used for the data bus. The next 3 or 7 pins (depending on bus width) are used automatically. Can use GP notation (e.g., `GP0`).
`clock_pin`	GPIO pin used for the clock signal. Must not overlap the 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.
`bus_width`	(Optional) Number of data pins: **8** (default) or **4**. Must be the same on both host and client.

MEMORY SHARE Command User Manual

Wiring

8-Bit Mode (Default)

The host and client are connected with 8 data wires, 1 clock wire, and 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

Total: 9 GPIO pins + ground on each board.

4-Bit Mode

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*clk*	GP*clk*	Clock
GND	GND	Ground

Total: 5 GPIO pins + ground on each board.

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 data pins.

- Keep wires short (under ~20 cm) for reliable high-speed operation. For longer wires, increase the clock divider.

- Both host and client must use the same bus width. There is no auto-detection.

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 on the lower half of the data pins and pulls the clock LOW. It then waits for the client to respond on the upper half. In 8-bit mode the host drives 0101 on pins 0-3 and reads pins 4-7. In 4-bit mode the host drives 01 on pins 0-1 and reads pins 2-3.

2. Client checks that the host clock is not already toggling. If it is, an error is raised. Otherwise, the client drives the complementary pattern on the upper data pins and waits to see the host's pattern on the lower pins.

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 clock cycle: one to output data (clock LOW) and one to hold (clock HIGH). Each clock cycle transfers one nibble (4-bit mode) or one byte (8-bit mode).

8-bit mode:

Clock Divider	Byte Rate (at 125 MHz)	Notes
10 (default)	3.125 MB/s	Conservative, reliable with breadboard wiring
4	7.8 MB/s	Recommended maximum for short direct connections
3 (minimum)	10.4 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          ' stops both host and client
MEMORY SHARE STOP HOST     ' stops host only
MEMORY SHARE STOP CLIENT   ' stops client only
```

Each variant stops the PIO state machine, aborts its DMA channels, resets all GPIO pins (data + clock) to their default state (SIO input, no pulls), releases pin reservations, and removes the PIO program.

- MEMORY SHARE STOP stops both the host and client on this board (if active).

- MEMORY SHARE STOP HOST stops only the local host, leaving a running client unaffected.

- MEMORY SHARE STOP CLIENT stops only the local client, leaving a running host unaffected.

It is safe to call any stop variant even when no share is active.

Running MEMORY SHARE HOST or MEMORY SHARE CLIENT a second time automatically stops the previous host or client (respectively) before starting the new one. The other role is not affected.

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.
`Bus width must be 4 or 8`	The `bus_width` parameter is not 4 or 8.
`Need N consecutive GPIOs from data pin`	The N pins starting at `data_pin` would exceed GPIO 29.

`Clock pin overlaps data pins`

The clock GPIO falls within the range of the 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.

Examples

8-Bit Mode (Default)

Host Program (RP2040)

```
' Host - transmits 100 integers to the client (8-bit bus)
Option Base 0
On Error Skip
Memory Share Stop
```

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

```
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, 8-bit
Memory Share Host 1, 0, GP0, GP8, addr%, 800, 10
```

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

```
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 (8-bit bus)
Option Base 0
On Error Skip
Memory Share Stop
```

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

```
Dim i%
For i% = 0 To 99
  shared%(i%) = -1
Next
```

```
' Start client: PIO 1, SM 0, data GP0, clock GP15, 800 bytes, 8-bit
Memory Share Client 1, 0, GP0, GP15, addr%, 800
```

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

```
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."
```

4-Bit Mode

Host Program (RP2040)

```
' Host - transmits using 4-bit bus (saves 4 GPIO pins)
Option Base 0
On Error Skip
Memory Share Stop
```

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

```
Dim i%, counter%
For i% = 0 To 99
  shared%(i%) = i%
Next
```

```
' Start host: PIO 1, SM 0, data GP0, clock GP4, 800 bytes, divider 10, 4-bit
Memory Share Host 1, 0, GP0, GP4, addr%, 800, 10, 4
```

```
Print "Host running (4-bit). Press any key to stop."
```

```
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 using 4-bit bus
Option Base 0
On Error Skip
Memory Share Stop
```

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

```
Dim i%
```

```
For i% = 0 To 99
  shared%(i%) = -1
Next
```

```
' Start client: PIO 1, SM 0, data GP0, clock GP15, 800 bytes, 4-bit
Memory Share Client 1, 0, GP0, GP15, addr%, 800, 4
```

```
Print "Client running (4-bit). Press any key to stop."
```

```
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."
```

Bidirectional (Simultaneous Host + Client)

A single PicoMite can act as both host and client at the same time, enabling two boards to exchange data in both directions. Each direction uses its own set of data pins, clock pin, and PIO state machine. This example uses 4-bit mode for both channels.

Wiring (10 signal wires + ground):

Board A Pin	Board B Pin	Function
GP0	GP0	Data bit 0 (A -> B)
GP1	GP1	Data bit 1 (A -> B)
GP2	GP2	Data bit 2 (A -> B)
GP3	GP3	Data bit 3 (A -> B)
GP22	GP22	Clock (A -> B)
GP4	GP4	Data bit 0 (B -> A)
GP5	GP5	Data bit 1 (B -> A)
GP6	GP6	Data bit 2 (B -> A)
GP7	GP7	Data bit 3 (B -> A)
GP26	GP26	Clock (B -> A)
GND	GND	Ground

Board A Program

```
' Board A: HOST on GP0-GP3/GP22 (sends), CLIENT on GP4-GP7/GP26 (receives)
Option Base 0
On Error Skip
Memory Share Stop
```

```
Dim tx%(99)
Dim txaddr% = Peek(VarAddr tx%())
Dim rx%(99)
Dim rxaddr% = Peek(VarAddr rx%())
```

```
Dim i%, counter%
For i% = 0 To 99
  tx%(i%) = i%
```

```
  rx%(i%) = -1
Next
```

```
' Start host (sending to Board B): PIO 1, SM 0, data GP0, clock GP22, 4-bit
Memory Share Host 1, 0, GP0, GP22, txaddr%, 800, 10, 4
```

```
' Start client (receiving from Board B): PIO 1, SM 1, data GP4, clock GP26, 4-bit
Memory Share Client 1, 1, GP4, GP26, rxaddr%, 800, 4
```

```
Print "Board A running. Press any key to stop."
```

```
counter% = 0
Do
  For i% = 0 To 99
    tx%(i%) = counter% + i%
  Next
  Print "TX=" Str$(counter%) "  RX[0]=" Hex$(rx%(0)) " RX[99]=" Hex$(rx%(99))
  counter% = counter% + 100
  Pause 500
Loop Until Inkey$ <> ""
```

```
Memory Share Stop
Print "Stopped."
```

Board B Program

```
' Board B: CLIENT on GP0-GP3/GP22 (receives), HOST on GP4-GP7/GP26 (sends)
Option Base 0
On Error Skip
Memory Share Stop
```

```
Dim rx%(99)
Dim rxaddr% = Peek(VarAddr rx%())
Dim tx%(99)
Dim txaddr% = Peek(VarAddr tx%())
```

```
Dim i%, counter%
For i% = 0 To 99
  tx%(i%) = 1000 + i%
  rx%(i%) = -1
Next
```

```
' Start client (receiving from Board A): PIO 1, SM 0, data GP0, clock GP22, 4-bit
Memory Share Client 1, 0, GP0, GP22, rxaddr%, 800, 4
```

```
' Start host (sending to Board A): PIO 1, SM 1, data GP4, clock GP26, 4-bit
Memory Share Host 1, 1, GP4, GP26, txaddr%, 800, 10, 4
```

```
Print "Board B running. Press any key to stop."
```

```
counter% = 0
Do
  For i% = 0 To 99
    tx%(i%) = 1000 + counter% + i%
  Next
```

```
  Print "TX=" Str$(1000 + counter%) "  RX[0]=" Hex$(rx%(0)) " RX[99]=" Hex$(rx%(99))
  counter% = counter% + 100
  Pause 500
Loop Until Inkey$ <> ""
```

```
Memory Share Stop
Print "Stopped."
```

Technical Details

Hardware Resources Used

Each active host uses:

- 1 PIO state machine (2 instruction slots)
- 2 DMA channels: SHARE_DMA_DATA (channel 10) and SHARE_DMA_CTRL (channel 11)
- 5 GPIO pins (4 data + 1 clock) in 4-bit mode, or 9 GPIO pins (8 data + 1 clock) in 8-bit mode

Each active client uses:

- 1 PIO state machine (3 instruction slots)
- 2 DMA channels: PIO_RX_DMA (channel 8) and PIO_RX_DMA2 (channel 9)
- 5 GPIO pins (4 data + 1 clock) in 4-bit mode, or 9 GPIO pins (8 data + 1 clock) in 8-bit mode

Because the host and client use different DMA channels, a single PicoMite can run one host and one client simultaneously for full-duplex (bidirectional) communication. The host and client must use different PIO state machines and non-overlapping GPIO pins.

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 in chunks of 4 or 8 bits on the data pins, toggling the clock via sideset. In 8-bit mode, each 32-bit word takes 4 clock cycles (4 bytes). In 4-bit mode, each 32-bit word takes 8 clock cycles (8 nibbles). 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 4 or 8 bits at a time. After accumulating 32 bits, autopush moves the assembled 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.

Since the host and client on each board maintain independent state, restarting one does not affect the other. For

example, MEMORY SHARE STOP HOST stops only the local host -- a running client on the same board continues uninterrupted.

For re-running a host/client pair across two boards, 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 to stop both roles, or MEMORY SHARE STOP HOST / MEMORY SHARE STOP CLIENT to stop individual roles. Running a new MEMORY SHARE HOST or CLIENT command automatically stops the previous instance of the same role before starting the new one.

Pin Constraints

- The data pins must be consecutive GPIOs (e.g., GP0-GP3 for 4-bit, GP0-GP7 for 8-bit).
- 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.
- Both host and client must specify the same bus width. Mismatched widths will result in corrupted data.