OSPI vs GPMC — Interfacing an FPGA at 64 Mbit/s (8 MB/s)

Assumption: sustained throughput target ≈ 8 MB/s. OSPI side assumes the FPGA mimics a NOR-like target; GPMC side assumes the FPGA exposes an async SRAM/peripheral window.

Dimension	OSPI (FPGA mimics NOR)	GPMC (FPGA as async SRAM/peripheral)
Basic model	Serial/QSPI/Octal master issues cmd+addr+dummy then reads data (**pull**).	Parallel host bus; SoC performs **read/write** cycles with CS#/OE#/WE# on 8/16-bit data.
Throughput headroom @ 8 MB/s	Easy if using **quad/octal** I/O or long **indirect/DMA** reads; single-lane must run high SCLK and still pays cmd/dummy tax.	Easy with modest async timings (e.g., **tACC ~50–80 ns**); 16-bit bus provides ample margin even with PIO.
Granularity efficiency	**Prefers big bursts**; tiny per-sample reads suffer (cmd/addr/dummy + driver setup).	**Good for small reads** (one bus cycle per word/halfword); no opcode overhead.
Push vs. pull	Always **pull** (SoC master).	SoC-initiated; supports both **read and write** bus cycles to FPGA logic.
CPU load (Linux A53)	Low if batched via **indirect + DMA** or **direct (XIP) + mem2mem DMA**;	Low–moderate with simple **PIO memcpy()** loops; GPMC DMA typically not used on

Dimension	OSPI (FPGA mimics NOR)	GPMC (FPGA as async SRAM/peripheral)
	high if many tiny reads.	AM62x Linux, but PIO still meets 8 MB/s.
Determinism / jitter	Clocking handled by controller; host jitter affects command scheduling. **Direct/XIP** risks speculation/prefetch quirks.	Predictable cycles; latency shaped by **tACC/WAIT**; easy to reason about per-sample pulls.
Prefetch/speculation risk	**Direct/XIP:** controller/CPU may prefetch beyond requested data (bad for live streams). **Indirect:** bounded by explicit length.	No controller prefetch; each read returns exactly the addressed data.
FPGA complexity	**Higher:** emulate NOR protocol (READ opcodes, dummy cycles, 3/4-byte addr, JEDEC/SFDP; optional DQS/training for DDR modes).	**Lower:** implement simple async memory/register interface; optional **WAIT** for back-pressure.
Pins	Fewer: **CS, SCLK, IO[0..3/7], (DQS)**.	More: **address + data (8/16) + CS#/OE#/WE#/WAIT**.
Read/Write support	Read-centric; writes require full flash program/erase semantics (usually avoided in stream windows).	Both **read & write** cycles supported natively (you choose what the FPGA implements).

Dimension	OSPI (FPGA mimics NOR)	GPMC (FPGA as async SRAM/peripheral)
Best use case	**Bulk pulls** from a linear window; long DMA reads; minimal pins.	**Per-sample or small-burst** access; simple control/status writes; deterministic timing.
Worst use case	**Tiny per-sample reads** at high rate; XIP on live data without guarding speculation.	**Extreme pin constraints**, or need for very long continuous bursts with minimal pins.
Back-pressure	Host simply **doesn't read**; add a sideband GPIO watermark if needed.	Use **WAIT** pin (or status register) to stall host during reads; straightforward flow control.
Linux driver effort	Use **spi-mem/OSPI** driver; for a 'NOR-lite' read-only window you still write a small driver to issue long indirect reads safely.	Simple **platform driver/UIO** to map GPMC window and `memcpy()`; no flash stacks required.
Integration risk	Protocol quirks (dummy, mode changes, JEDEC/SFDP); **XIP prefetch hazards** on dynamic data.	Signal integrity and timing (**tACC**, setup/hold) + more pins; otherwise straightforward.
Hitting 8 MB/s	**Yes**, via quad/octal or batched **indirect** reads (or **direct + DMA memcpy**).	**Yes**, even **PIO on 16-bit** with moderate timings; 8-bit also feasible with sane tACC.

Notes

- OSPI in **XIP/direct** mode may prefetch; for live streams, **indirect/DMA with large transfers** is safer.
- GPMC timing and **WAIT** shape latency; even **PIO** on a **16-bit** bus easily meets **8 MB/s**.