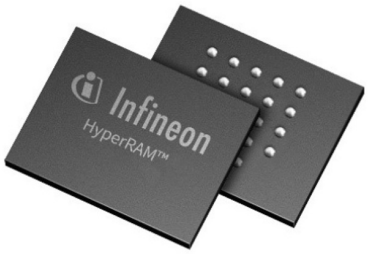
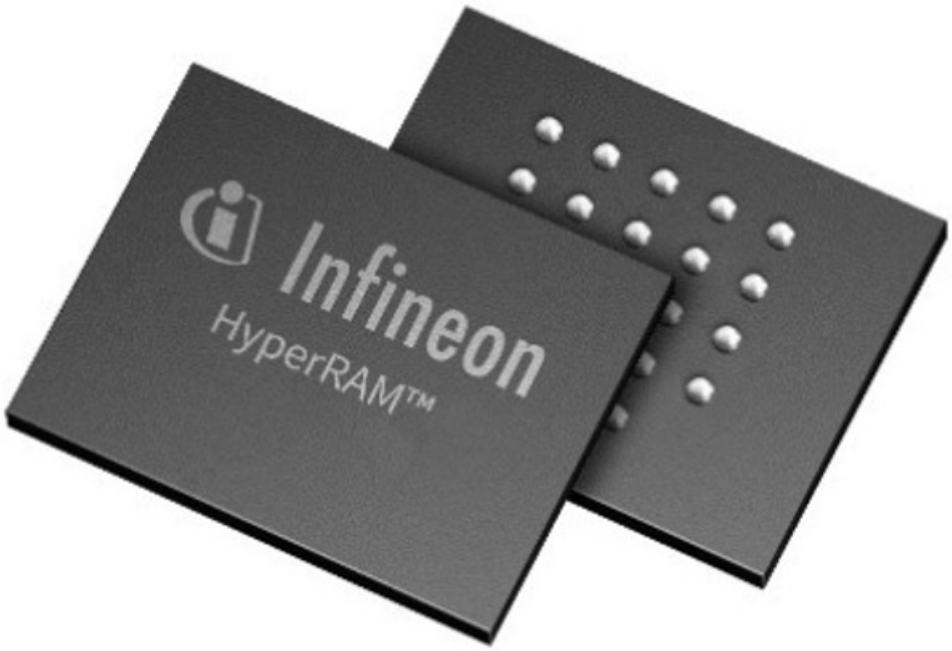


infineon AN226576 HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device User Guide

AN226576 HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device User Guide



Contents

1 AN226576 HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device
2 Introduction
3 HYPERBUS™ – a primer
4 Octal SPI (xSPI) – a primer
5 HYPERRAM™ product overview
6 Designing with HYPERRAM™
7 HYPERRAM™ – a low-pin-count, high-performance system memory
8 Revision history
9 Documents / Resources
10 Related Posts

AN226576 HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device

Getting started with HYPERRAM™
About this document
Scope and purpose
This application note gives an overview of critical concepts needed to design in with Infineon's latest high-density, high-performance memory, and lists the key advantages of
using HYPERRAM™ in a system and typical use case scenarios.
This application note also provides resources for designing in HYPERRAM™ with Infineon TRAVEO™ MCU as well as leading system-on-chip (SoC) in the market.
Intended audience
This document is primarily intended for anyone who wants to start with HYPERRAM™. Associated part family S27KL0641/ S27KL0642/ S27KL0643
Getting started with HYPERRAM™

Introduction

Introduction HYPERBUS™ and Octal SPI (CSPI) HYPERRAM™ are both high-performance 8-bit-wide serial self-refresh DRAM devices introduced by Infineon as a part of the wider HYPERBUS™/xSPI memory family.
The biggest advantage of serial interface memories is a reduction in the number of signals needed for interfacing the memory to a host controller, resulting in reduced package and multi-layer PCB costs. However, serial memories historically have lower data throughput or longer random-access time. This is mitigated in HYPERBUS™ and CSPI with high-performance multi-I/O DDR architecture where source-synchronous data is captured at twice the clock frequency rate.
First-generation HYPERRAM™ devices supported up to 166 MHz DDR clock frequency whereas second-generation devices that now support a clock frequency of 200 MHz have a peak data transfer rate of 0.4 Gbps.
Figure 1 provides a data throughput comparison across different interfaces.

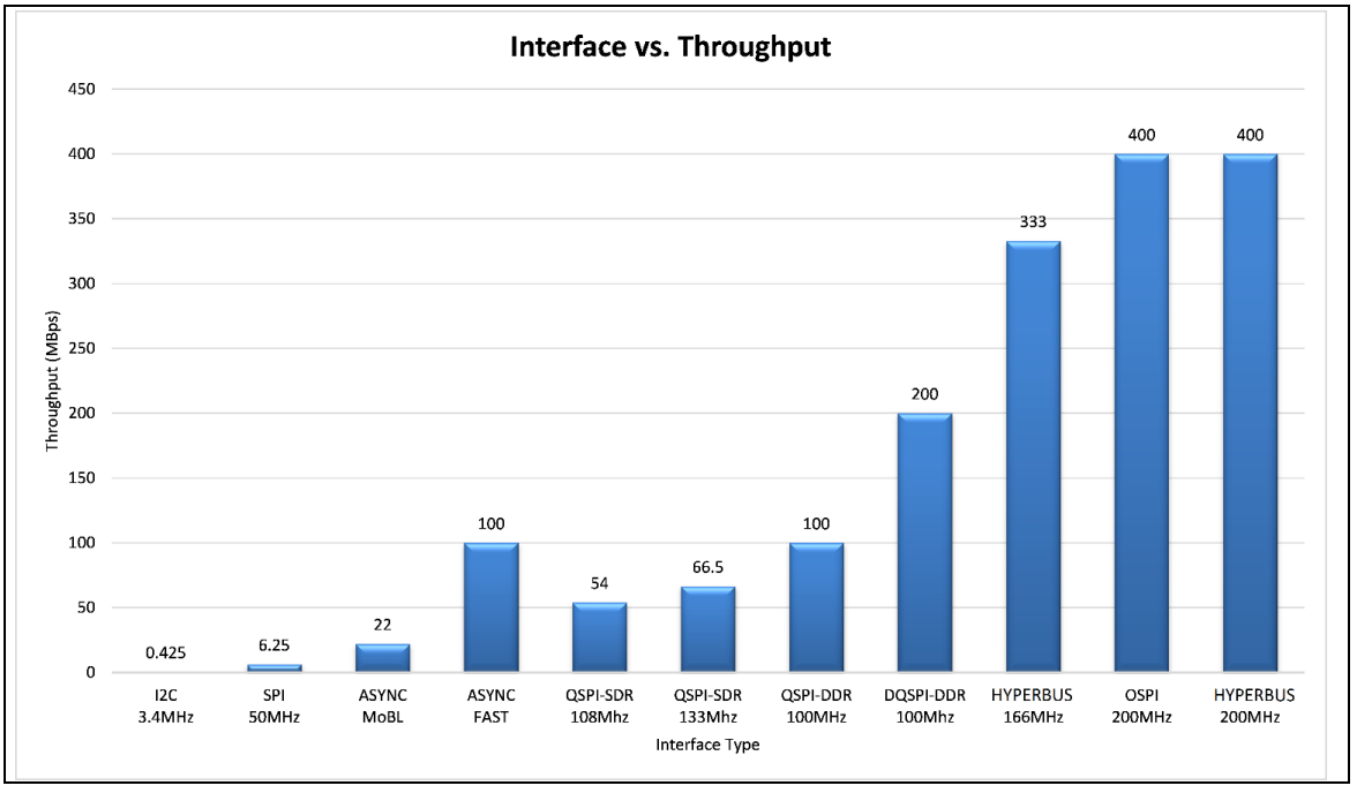


Figure 1 Interface data throughputs

HYPERBUS™ – a primer

HYPERBUS™ products use the high-performance HYPERBUS™ to connect between a host system Master and one or more Slave interfaces. HYPERBUS™ is used to connect microprocessor, microcontroller, or ASIC devices with random access NOR flash memory, RAM, or peripheral devices.

HYPERBUS™ is an interface that draws upon the legacy features of both parallel and serial interface memories while enhancing system performance, ease of design, and system cost reduction.

Parallel flash and PSRAM have long been the standard for simple interface, high performance, random access memory, used for embedded system code execution and data storage. However, parallel interface memory requires a high signal count with separate control, address, and data connections that involve 45 or more signals. There have been parallel interface variations that reduce the signal count by multiplexing some address and data signals yet, may still involve 20 or more signals. These high signal counts provide high data throughput but at the cost of many connectors on the host system processor or ASIC and, higher-cost multilayer PCBs that suffer from signal routing congestion.

Many systems have moved to serial interface memories to reduce the number of signals needed for the connection to memory, thereby freeing up host system connections for use by other features reducing the package and multi-layer PCB cost. However, serial memories generally have lower data throughput or longer random-access time, which may relegate the role of serial memories to just transferring code and data to DRAM memory for random access and high throughput.

HYPERBUS™ has a low signal count, and double data rate (DDR) interface that achieves high read and writes throughput while reducing the number of device I/O connections and signal routing congestion in a system.

These devices transmit data and command/address information in DDR mode over the 8-bit data bus. The clock input signals are used for signal capture by devices when receiving command/address/data information.

Data Strobe (RWDS/DQS), which is an output in both interfaces, indicates when data is being transferred out of the devices to the host. RWDS/DQS is referenced to the rising and falling edges of the clock during the data transfer portion of reading operations.

Command, address, and data information is transferred over the eight HYPERBUS™ DQ[7:0] signals. The clock input (CK#, CK) is used for information capture by an HYPERBUS™ Slave device when receiving the command, address, or data on the DQ signals. Command/address/write-data values are center-aligned with the clock edges, whereas read-data values are edge-aligned with the transitions of RWDS/DQS.

All HYPERBUS™ and OSPI inputs/outputs are LVCMOS-compatible, supporting either 1.8 V or 3.0 V (nominal) voltage supplies. Control signals are all single-ended except for the master clock. OSPI's master clock is single-ended whereas HYPERBUS™ requires the master clock to be differential for the 1.8-V architecture only.

HYPERBUS™ and OSPI instruction protocols follow the traditional industry-standard Serial Peripheral

Interface (SPI). Every transaction begins with the assertion of Chip Select (CS#). This is followed by the transfer of Command and Address bytes with access latency and either read or write data transfers until CS# is deasserted.

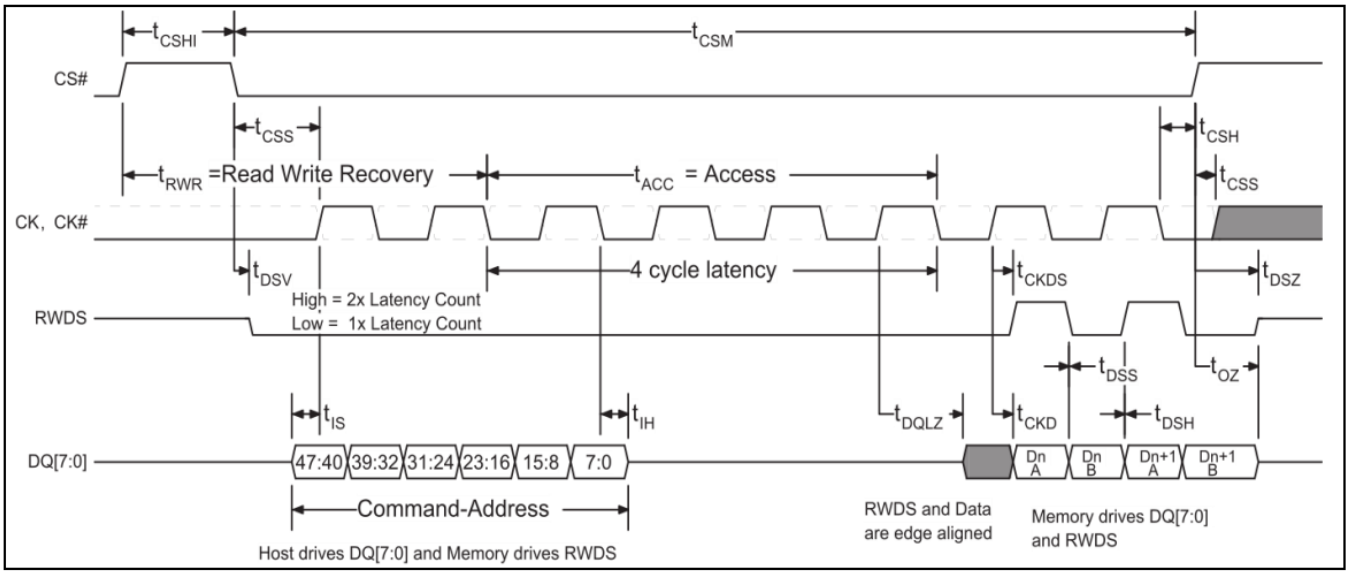


Figure 2 Read transaction, single initial latency count

The read/write data strobe (RWDS) is a bidirectional signal that indicates:

- When data will start to transfer from an HYPERRAM™ device to the Master device in reading transactions (initial read latency)
- When data is being transferred from an HYPERRAM™ device to the Master device during reading transactions (as a source-synchronous read data strobe)
- When data may start to transfer from the Master device to an HYPERRAM™ device in write transactions (initial write latency)
- Data masking during write data transfers.

During the CA transfer portion of a read or write transaction, RWDS acts as an output from an HYPERRAM™ device to indicate whether additional initial access latency is needed in the transaction.
During read data transfers, RWDS is a read data strobe with data values edge aligned with the transitions of RWDS.

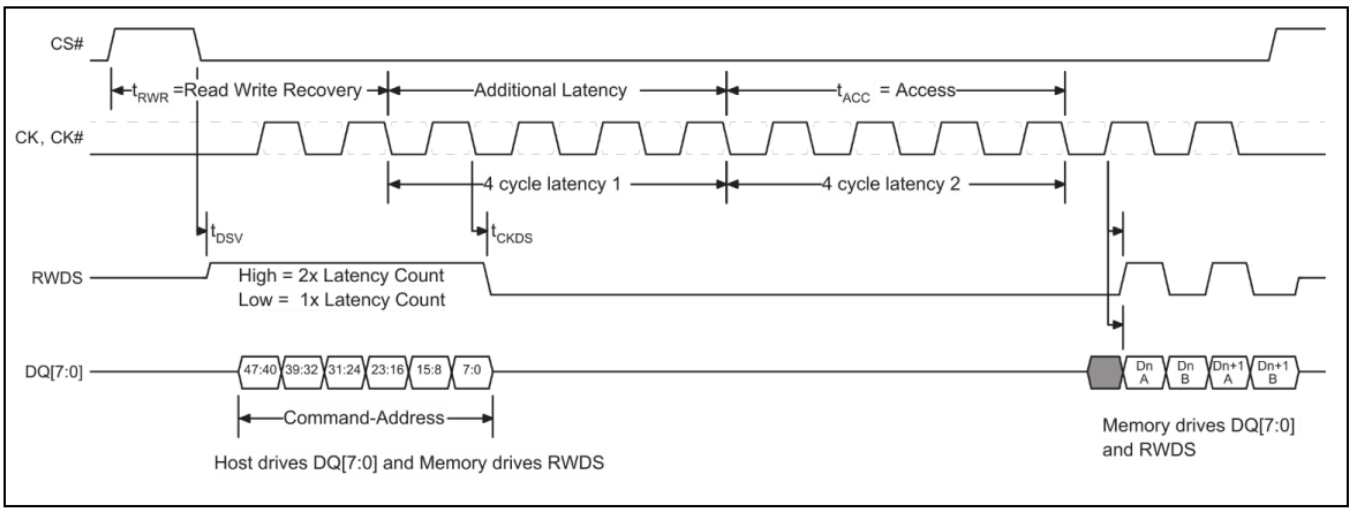


Figure 3 Read transaction, additional latency count

HYPERBUS™ – a primer

During write data transfers, RWDS indicates whether each data byte transfer is masked with RWDS HIGH (invalid and prevented from changing the byte location in memory) or not masked with RWDS LOW (valid and written to memory). Data masking may be used by the host to byte-align the write data within a memory or enable the

merging of multiple non-word-aligned writes in a single burst write. During write transactions, data is center-aligned with clock transitions.
Refer to Datasheet for a detailed description of the HYPERBUS™ protocol and relevant timing parameters.

Octal SPI (xSPI) – a primer

xSPI (Octal) is an SPI-compatible, low-signal-count, Double Data Rate (DDR) interface supporting eight I/Os. The DDR protocol in xSPI (Octal) transfers two data bytes per clock cycle on the DQ input/output signals. A read or write transaction on xSPI (Octal) consists of a series of 16-bit-wide, one-clock-cycle data transfers at the internal RAM array with two corresponding 8-bit-wide, one-half-clock-cycle data transfers on the DQ signals.
xSPI has been adopted as one of the JEDEC standards for (JESD251) for external memory interface and has seen wide adoption by microcontrollers and IP vendors in the market.
Each transaction on xSPI (Octal) must include a command whereas address and data are optional. The transactions are structured as follows:

- Each transaction begins with CS# going LOW and ends with CS# returning HIGH.
- The serial clock (CK) marks the transfer of each bit or group of bits between the host and memory. All transfers occur on every CK edge (DDR mode).
- Each transaction has a 16-bit command that selects the type of device operation to perform.

Note:
The 16-bit command is based on two 8-bit opcodes. The same 8-bit opcode is sent on both edges of the clock.

- A command may be standalone or may be followed by address bits to select a memory location in the device to access data.
- Read transactions require a latency period after the address bits and can be zero to several CK cycles. CK must continue to toggle during any read transaction latency period.

Note:
During the command and address parts of a transaction, the memory can indicate whether an additional latency period is needed for a required refresh time (truth) which is added to the initial latency period; by driving the RWDS signal HIGH.

- Write transactions to registers do not require a latency period.
- Write transactions to the memory array require a latency period after the address bits and can be zero to several CK cycles. CK must continue to toggle during any write transaction latency period.

Note:
During the command and address parts of a transaction, the memory can indicate whether an additional latency period is needed for a required refresh time (truth), which is added to the initial latency period by driving the RWDS signal HIGH.

- In all transactions, command and address bits are shifted in the device with the most significant bits (MSB) first. Individual data bits within a data byte are shifted in and out of the device MSb first as well. All data bytes are transferred with the lowest address byte sent out first.

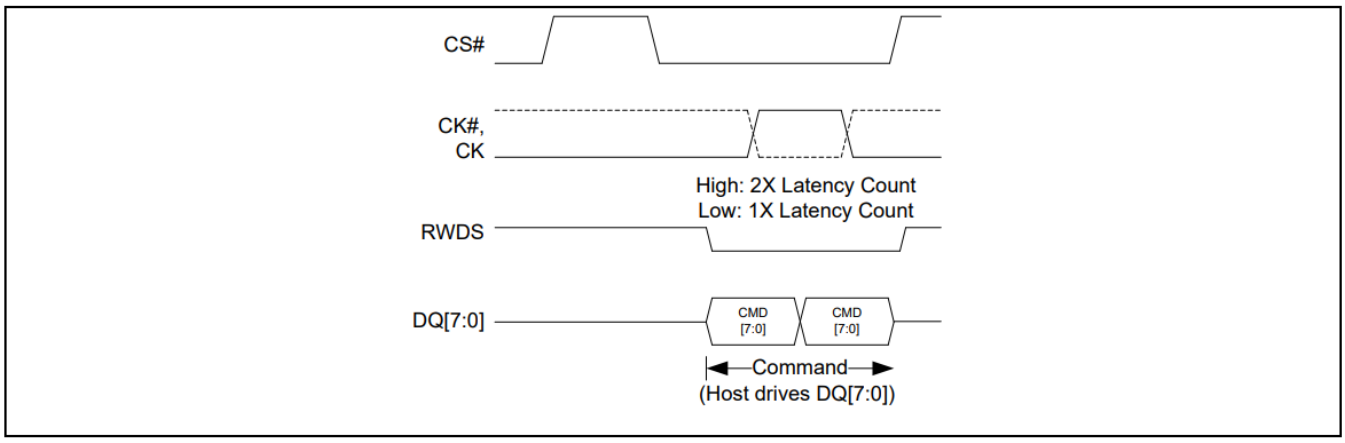


Figure 4 xSPI (Octal) command-only transaction (DDR)

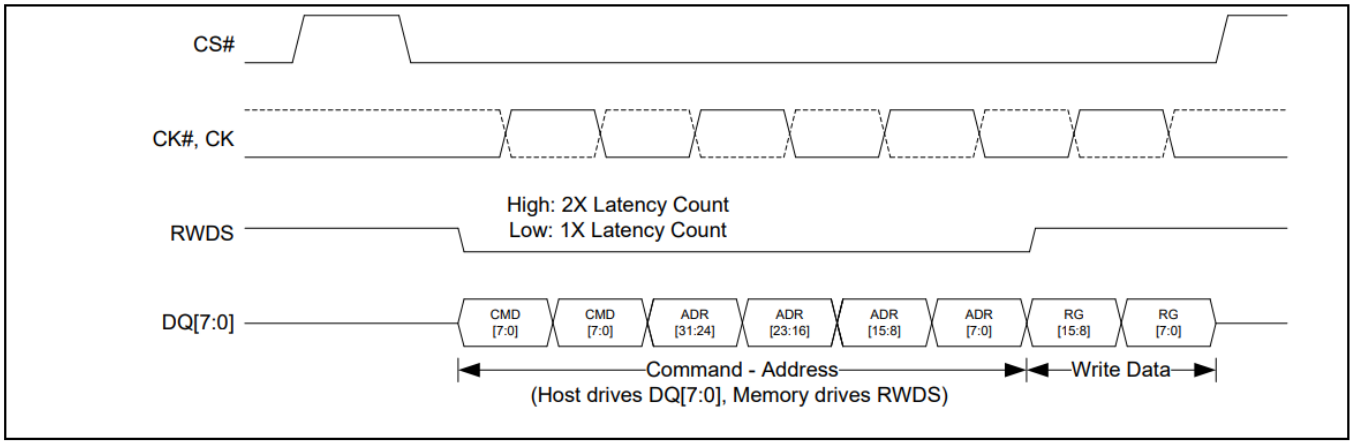


Figure 5 xSPI (Octal) write with no latency transaction (DDR) (register writes)

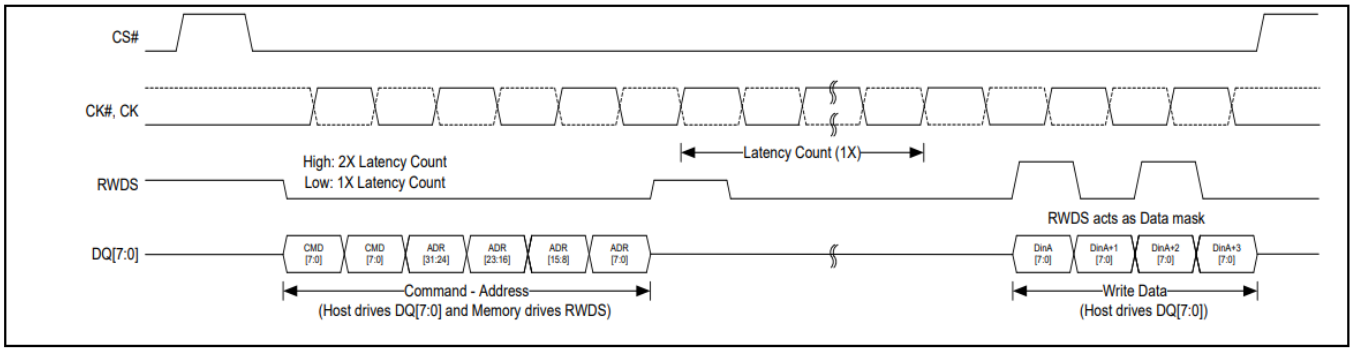


Figure 6 xSPI (Octal) write with 1x latency transaction (DDR) (memory array writes)

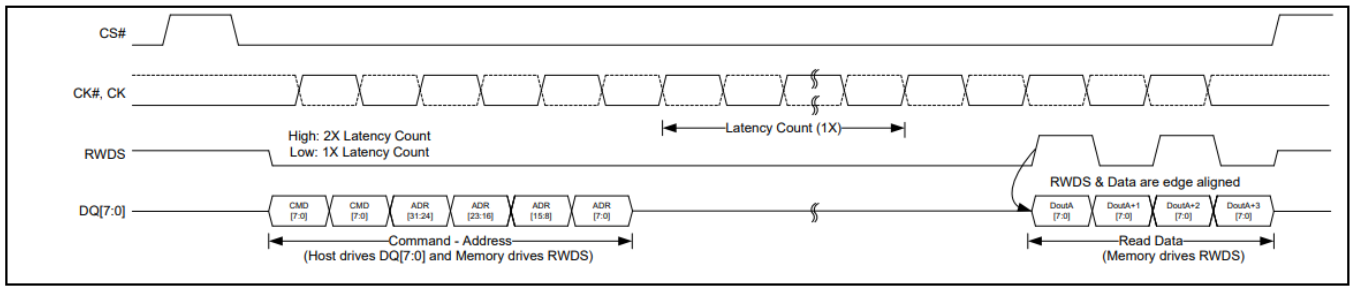


Figure 7 xSPI (Octal) read with 1x latency transaction (DDR) (all reads)

xSPI as a standard is supported by several Infineon Flash devices. However, the HYPERRAM™ device covered in this document supports only the (8-8-8) format of operation. Every single opcode, address, and data has to be sent out on 8 I/Os. Refer to the CSPI HYPERRAM™ datasheet for a detailed description of CSPI protocol and relevant timing parameters.

HYPERRAM™ product overview

The 64-Mb HYPERRAM™ device is 1.8 V or 3.0 V array and I/O, synchronous self-refresh dynamic RAM (DRAM).

The HYPERRAM™ device provides an HYPERBUS™ interface to the host system. 2nd generation HYPERRAM™ also added the support of xSPI interface. Both interfaces have an 8-bit (1 byte) wide DDR data bus and use only word-wide (16-bit data) address boundaries. Read transactions provide 16 bits of data during each clock cycle (8 bits on both clock edges). Write transactions take 16 bits of data from each clock cycle (8 bits on each clock edge)

Read and write transactions require three clock cycles to define the target row/column address and then an initial access latency of that. During the Command (CA) part of a transaction, the memory will indicate whether an additional latency for a required refresh time (truth) is added to the initial latency by driving the RWDS signal HIGH. During a read (or write) transaction, after the initial data value has been output (or input), additional data can be read from (or written to) the row on subsequent clock cycles in either a wrapped or linear sequence.

When configured in linear burst mode, the device will automatically fetch the next sequential row from the memory array to support a continuous linear burst. Simultaneously accessing the next row in the array while the read or write data transfer is in progress allows for a linear sequential burst operation that can provide a sustained data rate of 400 MB/s (1 byte (8-bit data bus) * 2 (data clock edges) * 200 MHz = 400 MB/s) for 2nd Generation HYPERRAM™ whereas 333 MB/s for 1st Generation HYPERRAM™ device.

4.1 Signal description

Both HYPERBUS™ and xSPI (Octal) interfaces have identical signal naming conventions. Table 1 below summarizes all the signals of an HYPERRAM™ device.

Table 1

Signal description summary

Pin name	Type	Description
CS#	Input	Chip select. Bus transactions are initiated with a HIGH-to-LOW transition. Bus transactions are terminated with a LOW-to-HIGH transition. The master device has a separate CS# for each Slave.
CK, CK#	Input	Differential clock. Command, address, and data information is output with respect to the crossing of the CK and CK# signals. A differential clock is optional on 1.8 V / 3.0 V I/O devices. Single-ended clock. CK# is not used, only a single-ended CK is used. The clock is not required to be free-running.
DQ[7:0]	Input / output	Data input/output. Command, address, and data information is transferred on these signals during reading and write transactions.
RWDS	Input / output	Read data strobe. During the command/address portion of all bus transactions, RWDS is a slave output and indicates whether additional initial latency is required. slave output during reading data transfer, data is edge aligned with RWDS. Slave input during data transfer in write transactions to function as a data mask. (HIGH = additional latency, LOW = no additional latency).
RESET#	Input	Hardware RESET. When LOW, the slave device will self-initialize and return to the standby state. RWDS and DQ[7:0] are placed into the High-Z state when the RESET# is LOW. The slave RESET# input includes a weak pull-up, if RESET# is left unconnected it will be pulled up to the HIGH state.

VCC	Power supply	Array power
CQC	Power supply	Input / output power
GND VSS	Power supply	Array ground
VSSQ	Power Sup ply	Input / output ground
RFU	No, connect	Reserved for future use. It May or may not be connected internally, the signal/ball loc ation should be left unconnected and unused by the PCB routing channel for future c ompatibility. The signal/ball may be used by a signal in the future.

4.2 Typical system connection

A typical HYPERBUS™ system can have multiple Slave memories connected to a Master through separate chip select signals (CS). Figure 8 shows a typical configuration. One of the Slaves can be an HYPERFLASH™ device while the other can be an HYPERRAM™ device.

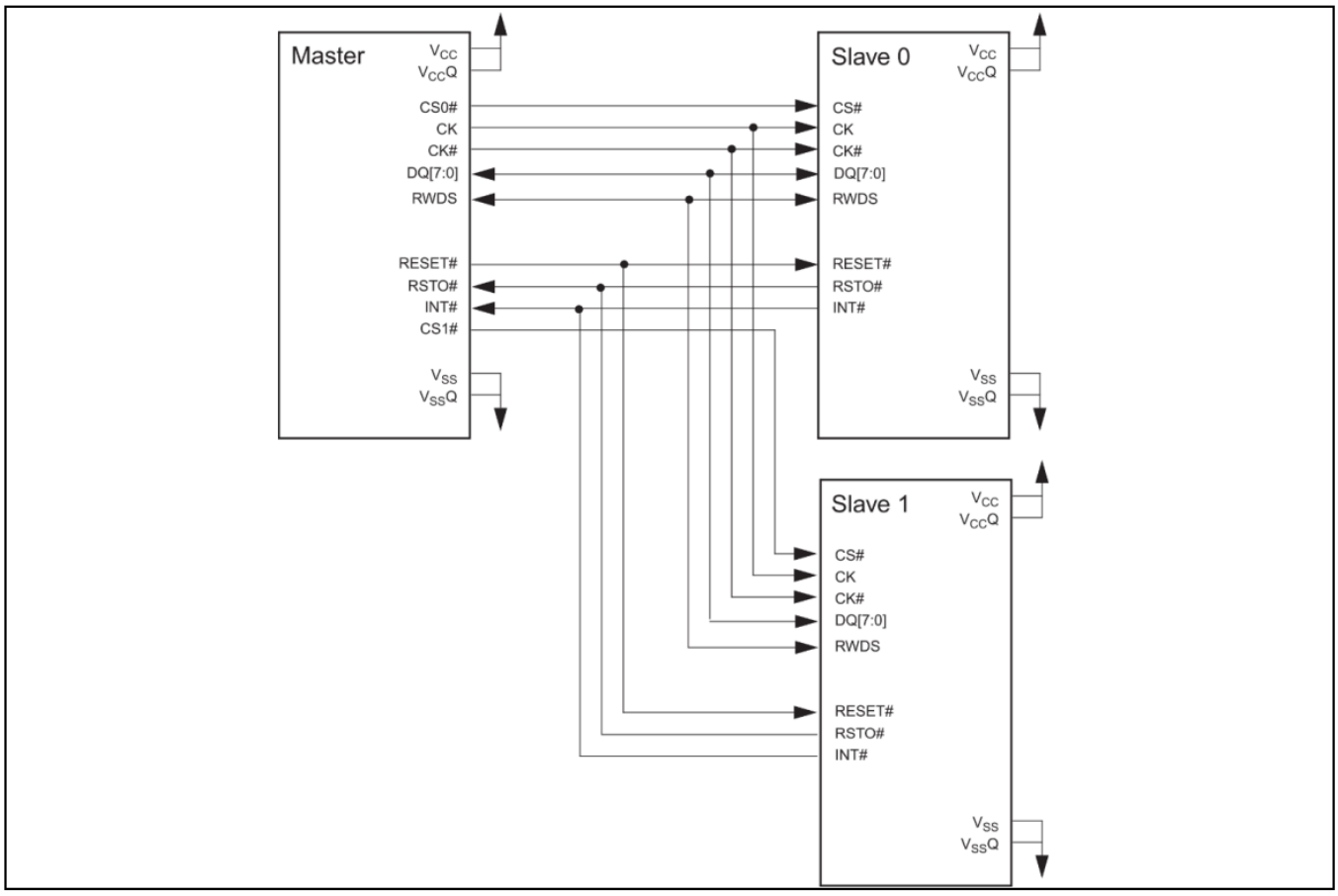


Figure 8 A typical system configuration

The master can be a microcontroller that supports HYPERBUS™ interface or an IP implemented in an SoC. As HYPERRAM™ is inherently a DRAM device, the cells need periodic refresh cycles to retain the data. This is addressed by internal self-refresh logic called distributed refresh logic.

4.3 Distributed refresh logic

The DRAM array requires a periodic refresh of all bits in the array. This can be done by the host system by reading or writing a location in each row within a specified time limit. The read or write access copies a row of bits to an internal buffer. At the end of the access the bits in the buffer are written back to the row in memory, thereby recharging (refreshing) the bits in the row of DRAM memory cells.

HYPERRAM™ devices include self-refresh logic that will refresh rows automatically. The automatic refresh of a row can only be done when the memory is not being actively read or written by the host system. The refresh logic

waits for the end of any active read or write before doing a refresh if a refresh is needed at that time. If a new read or write begins before the refresh is completed, the memory will drive RWDS HIGH during the CA period to indicate that an additional initial latency time is required at the start of the new access in order to allow the refresh operation to complete before starting the new access.
The required refresh interval for the entire memory array varies with temperature as shown in **Table 2**. This is the time within which all rows must be refreshed. Refresh of all rows could be done as a single batch of accesses at the beginning of each interval, in groups (burst refresh) of several rows at a time, spread throughout each interval, or as single-row refreshes evenly distributed throughout the interval. The self-refresh logic distributes single-row refresh operations throughout the interval so that the memory is not busy doing a burst of refresh operations for a long period, such that the burst refresh would delay host access for a long period

Table 2 Array refresh interval per temperature

Device temperature (°C)	Array refresh interval (ms)	Array rows	Recommended CSM (µs)
85	64	8192	4
105	16	8192	1

The distributed refresh method requires that the host does not do burst transactions that are so long as to prevent the memory from doing the distributed refreshes when they are needed. This sets an upper limit on the length of reading and writing transactions so that the refresh logic can insert a refresh between transactions. This limit is called the CS# LOW maximum time (t_{CSM}). The t_{CSM} value is determined by the array refresh interval divided by the number of rows in the array, then reducing this calculation by half to ensure that a distributed refresh interval cannot be entirely missed by a maximum length host access starting immediately before a distributed refresh is needed. Because t_{CSM} is set to half the required distributed refresh interval, any series of maximum-length host accesses that delay refresh operations will catch up on refresh operations at twice the rate required by the refresh interval divided by the number of rows.
The host system is required to respect the t_{CSM} value by ending each transaction before violating t_{CSM}. This can be done by host memory controller logic splitting long transactions when reaching the t_{CSM} limit, or by host system hardware or software not performing a single read or write transaction that would be longer than t_{CSM}.
As noted in Table 2, the array refresh interval is longer at lower temperatures such that t_{CSM} could be increased to allow longer transactions. The host system can either use the t_{CSM} value from the table for the maximum operating temperature or, may determine the current operating temperature from a temperature sensor in the system in order to set a longer distributed refresh interval.

4.4 Power modes

Both HYPERBUS™ and xSPI HYPERRAM™ devices support the following identical power modes.
Active mode: The normal operating mode used for accessing the HYPERRAM™ is defined as active mode. Enabling CS# signal initiates device active mode. Every single device operation has to be performed in active mode.
Interface standby: Standby is the default, low-power state for the interface while the device is not selected by the host for data transfer (CS#= HIGH). All inputs, and outputs other than CS# and RESET# are ignored in this state.
Active clock stop: The active clock stop state reduces device interface energy consumption to the I_{CC6} level during the data transfer portion of a read or writes operation. The device automatically enables this state when the clock remains stable for t_{ACC} + 30 ns. While in the active clock stop state, read data is latched and always driven onto the data bus.
Active clock stop state helps reduce current consumption when the host system clock has stopped to pause the data transfer. Even though CS# may be LOW throughout these extended data transfer cycles, the memory device host interface will go into the Active Clock Stop current level at t_{ACC} + 30 ns. This allows the device to transition into a lower current state if the data transfer is stalled. Active read or write current will resume once the data transfer is restarted with a toggling clock. The active clock stop state must not be used in violation of the t_{CSM} limit. CS# must go HIGH before t_{CSM} is violated. The clock can be stopped during any portion of the active transaction as long as it is in a LOW state. Note that it is recommended to avoid stopping the clock during register access. Refer to the device datasheet for detailed specifications.
Deep power down: In the deep power down (DPD) state, current consumption is driven to the lowest possible level (PD). DPD state is entered by writing a '0' to the relevant bit in the device configuration register. The device reduces power within t_{DPDIN} time and all refresh operations stop. The data in memory space is lost, (becomes invalid without refresh) during DPD state. Driving CS# LOW then HIGH will cause the device to exit DPD state.
Also, POR or a hardware reset will cause the device to exit DPD state. Returning to a standby state requires

t_{EXTDPD} time. Returning to a standby state following a POR requires t_{VCS} time, as with any other POR. Following the exit from DPD due to any of these events, the device is in the same state as the following POR.
Note: In xSPI (Octal), deep power down transactions or writing any registered transaction can be used to enter DPD.
Hybrid sleep: This power is available only in the 2nd generation of the HYPERRAM™ device. In the hybrid sleep (HS) state, the current consumption is reduced (i_{HS}). HS state is entered by writing a '0' to the relevant bit in the device configuration register. The device reduces power within t_{HSIN} time. The data in memory space and register space is retained during the HS state. Bringing CS# LOW will cause the device to exit the HS state and a relevant bit to '1'. Also, POR or a hardware reset will cause the device to exit the hybrid sleep state.
Note that a POR or a hardware reset disables refresh where the memory core data can potentially get lost. Returning to a standby state requires t_{EXITHS} time. Following the exit from HS due to any of these events, the device is in the same state as entering hybrid sleep. Refer to datasheet of 2nd generation HYPERRAM™ device for details on hybrid sleep power mode.

Hardware reset: The RESET# input provides a hardware method of returning the device to the standby state. During trips, the device will draw an I_{CC5} current. If RESET# continues to be held LOW beyond t_{RPH}, the device draws a CMOS standby current (I_{CC4}). While RESET# is LOW (during t_{RP}), and during trips, bus transactions are not allowed.
A hardware reset will do the following:

- Cause the configuration registers to return to their default values
- Halt self-refresh operation while RESET# is LOW – memory array data is considered as invalid
- Force the device to exit the hybrid sleep state
- Force the device to exit deep power down state

After RESET# returns HIGH, the self-refresh operation will resume. Because the self-refresh operation is stopped during RESET# LOW, and the self-refresh row counter is reset to its default value, some rows may not be refreshed within the required array refresh interval per **Table 2**. This may result in the loss of DRAM array data during or immediately following a hardware reset. The host system should assume DRAM array data is lost after a hardware reset and reload any required data.
Software reset: The software reset provides a software method of returning the device to the standby state. During t_{SR}, the device will draw an I_{CC5} current. A software reset will do the following:

- Cause the configuration registers to return to their default values
- Halt self-refresh operation during the software reset process – memory array data is considered as invalid

After the software reset finishes, the self-refresh operation will resume. Because the self-refresh operation is stopped, and the self-refresh row counter is reset to its default value, some rows may not be refreshed within the required array refresh interval per **Table 2**. This may result in the loss of DRAM array data during or immediately following a software reset. The host system should assume DRAM array data is lost after a software reset and reload any required data.
Refer to application note AN226137 – Migrating from S27KS0641 to S27KS0642 for details on differences between 2nd Generation of HYPERRAM™ and 1st generation of HYPERRAM™ device.

Designing with HYPERRAM™

HYPERBUS™ and xSPI standards are widely getting adopted across the industry as one of the leading low pin count, high throughput interfaces. Infineon works closely with HYPERBUS™ chipset parts to enable HYPERBUS™ to interface as a hard IP in SoC or through Soft IP on leading FPGA vendors.
Table 3 HYPERBUS™ chipset support

Partner	Chipset / platform name	Application	HYPERFLASH™	HYPERRAM™
Infineon	TRAVEO™ S6J331x	Automotive Cluster	—	—
	TRAVEO™ S6J335x	Automotive Gateway	—	—
	TRAVEO™ S6J326Cx	Automotive Cluster	—	—
	TRAVEO™ S6J324Cx	Automotive Cluster	—	—
	TRAVEO™ S6J327Cx	Automotive Cluster	—	—
	TRAVEO™ S6J328Cx	Automotive Cluster	—	—
	TRAVEO™ S6J32DAx	Automotive Cluster	—	—
	TRAVEO™ S6J32BAx	Automotive Cluster	—	—
	FM4 family S6E2DH	Industrial	—	—
	series			
Altera/Intel	MAX10	Industrial	—	—
	Cyclone 10 LP	Industrial	—	—
GCT	GDM7243i	IoT	—	—
Greewaves Technologies	GAP8	Artificial Intelligence(AI), IoT	—	—
Maxim	MAX32650	Industrial, Portable Medical, IoT	—	—
NXP	**MAC57D5xxx**	Automotive Cluster	—	—
	S32K148	Automotive Generic / Body	—	—
	S32V23x	Automotive ADAS	—	—
	Kinetis K80	Industrial	—	—
	Kinetis K82	Industrial	—	—
	Kinetis K28F	Industrial	—	—
	i.MX8 family	Automotive Infotainment,	—	—
	i.MX RT1050	Industrial, Consumer	—	—
	i.MX RT1020	Industrial	—	—
	i.MX RT1060	Industrial	—	—
	i.MX RT106A	Industrial	—	—
	I.MX RT family	MCU-Based Solution for Alexa Voice Service	—	—

		NXP app note to support		
		HYPERRAM™ on I.MX RT family		
Renesas	R-CAR D3	Automotive Cluster	—	—
	R-Car H3	Automotive Infotainment / ADAS	—	—
	R-CAR M3	Automotive Infotainment / ADAS	—	—
	R-Car V3M	Automotive ADAS	—	—
	RZ/A2M	Industrial, AI	—	—

ST	**STM32L4Rx** Chorus SPC58 H	Industrial Automotive Gateway	— —	— —
Texas Instruments	Sitara AM6xxx	Industrial, Networking	—	—
Xilinx	Virtex UltraScale+ (xilinx.com) KINTEX UltraScale+ (16nm) VIRTEX UltraScale (20nm) KINTEX UltraScale (20nm) VIRTEX-7 (28nm) KINTEX-7 (28nm) Zynq 7000 Zynq UltraScale+ Artix-7	Communications Communications Communications Communications / Industrial Networking Networking Various Industrial Industrial	— — — — — — — — —	— — — — — — — — —

Table 4 HYPERBUS™ third-party development platform

Partner	Third-party development platform	Application	HYPERFLASH™	HYPERRAM™
Trenz Electronics	TE0725 with Xilinx Artix-7 TE0748 with Xilinx Artix	Industrial Secure SD	— —	— —
Dev boards	HyperMAX with Altera MAX10	Industrial, Medical, Automotive	—	—

Table 5 HYPERBUS™ memory controller IP

IP supplier	Link	Can be integrated in FPGA as soft IP	Can be integrated in a SoC
Infineon	Link Infineon	—	—
Cadence	Link Cadence	—	—
Mobiveil	Link Mobiveil	—	—
Synaptic Laboratories Ltd.	Link Synaptic Labs	—	—

Table 6 HYPERBUS™ memory controller verification IP

IP supplier	Link	HYPERFLASH™	HYPERRAM™
Cadence	Link for Cadence	—	—

Infineon Technologies also provides several design (IBIS and behavioral) models that allow customers to get a head start on system design. For the latest update on chipset pairing and support, see the Infineon HYPERBUS™ Ecosystem webpage.

HYPERRAM™ – a low-pin-count, high-performance system memory

The size of internal RAM is one of the most important factors when determining how to execute complex algorithms with a controller or an FPGA/SoC. Adequate RAM or system memory allows code to be fetched and executed by the processor. This RAM may also double as Stack/Heap storage to be utilized by application code to implement multiple levels of In-Service Routines that need context and state storage space.
With the growing complexity of applications, controllers/ SoCs quickly run out of system memory. Designers frequently solve this problem by adding fast external RAM using SDRAM or DDR RAM. This scalability means the designer can add several megabytes of high-performance RAM without compromising algorithmic complexity. However, SDRAM and DDR memory have serious drawbacks. SDRAM/DDR interfaces use a significant number of I/Os, requiring a dedicated interface while increasing PCB design and fabrication costs.
A recent trend in SoC/Controller design has seen the adoption of HYPERBUS™ as a default standard for NOR Flash memory. Current HYPERFLASH™ devices can deliver up to 400 MBps throughput while running the interface at 200 MHz DDR, requiring only 12 signals compared to 30-35 I/Os needed for SDRAM interfaces. HYPERRAM™ uses the same high-performance interface as HYPERFLASH™ but operates as true expansion RAM. Therefore, when an SoC/Controller/FPGA is running short of internal memory, you can use this high-performance, low-pin-count system memory while reusing an existing HYPERBUS™ interface. HYPERRAM™ is widely used in Automotive Clusters to store high-resolution display elements that need frequent updates. In industrial environments, controllers can increase working memory by orders of magnitude by using HYPERRAM™.

Figure 9 and Figure 10 show usage models of HYPERRAM™ in systems. Automotive clusters perform the blending of several graphics layers to generate the final display in a car. The quality and complexity of the graphics depending on the size of the display buffer that can be implemented in the controller. Adding a high-performance HYPERRAM™ to such systems enables loading and generating several complex display elements without having any penalty on the refresh rate of the display.

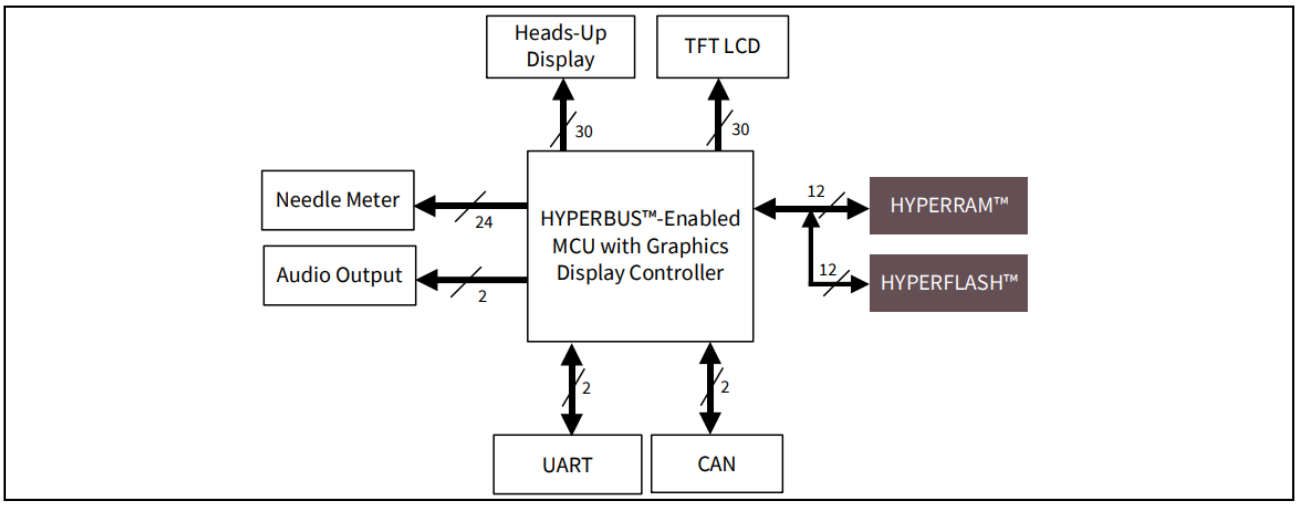


Figure 9 HYPERRAM™ as Dbuffer in automotive cluster

Another usage model for HYPERRAM™ is as an expansion memory for FPGA-based machine vision systems. A typical FPGA has limited RAM resources. These resources are better used for performing critical processing needed for Imaging algorithms. The configurations and code for the FPGA get loaded from the on-board flash or SD card onto the HYPERRAM™ for highest performance.

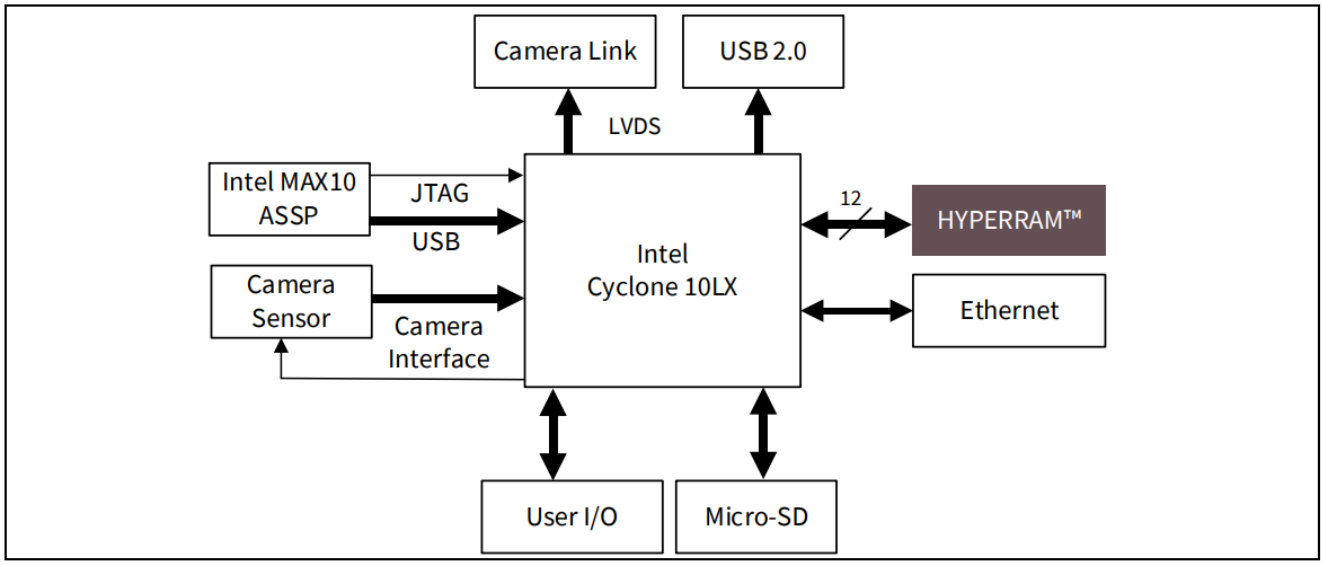


Figure 10 HYPERRAM™ as expansion memory in machine vision system

References

1. AN211622 – HYPERFLASH™ and HYPERRAM™ layout guide
2. AN209853 – HYPERRAM™ refresh interval optimization
3. AN218684 – HYPERBUS™ memory: Guide to efficient data access
4. AN226137 – Migrating from S27KS0641 to S27KS0642
5. HYPERBUS™ specification low signal count, high-performance DDR bus

Revision history

Document version	Date of release	Description of changes
**	2019-06-14	New application note
*A	2021-09-21	Migrated to Infineon template.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

IMPORTANT NOTICE
The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality condition or quality of the product. Before implementation of the product, the recipient of this
application note must verify any function and other technical information given herein in the reapplication. Infineon Technologies hereby disclaim any and all warranties and liabilities of any kind(including without limitation warranties of noninfringement of intellectual property rights of any third party) with respect to any and all information given in this application note.
The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of the customer's technical department to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.
For further information on the product, technology, delivery terms and conditions, and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.
Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon
Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.

Edition 2021-09-21
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to **www.cypress.com/support**
Document reference
002-26576 Rev. *A

Documents / Resources

	infineon AN226576 HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device [pdf] User Guide AN226576, HYPERBUS, 8 Bit Wide Serial Self Refresh DRAM Device, HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device, AN226576 HYPERBUS 8 Bit Wide Serial Self Refresh DRAM Device, Self Refresh DRAM Device, DRAM Device, S27KL0641, S27KL0642, S27KL0643