AN226576

Getting Started with HyperRAM™

Author: Nilesh Badodekar

Associated Part Family: S27KL0641/ S27KL0642/ S27KL0643

Related Application Notes: see Related Documents

AN226576 gives an overview of critical concepts needed to design in with Cypress' 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 Cypress' Traveo™ MCU as well as leading SoC in market.

Contents

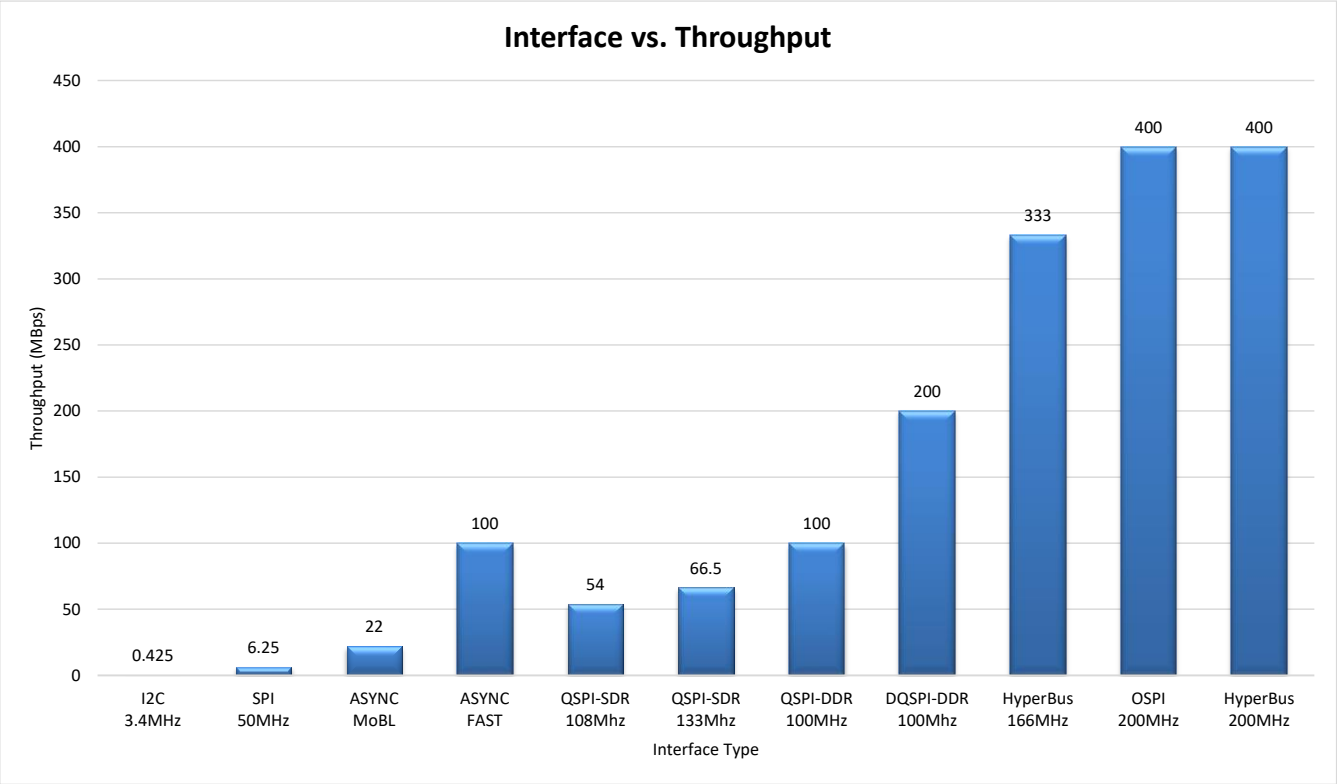
1 Introduction

HyperBus™ and Octal SPI (xSPI) HyperRAM™ are both high-performance 8-bit-wide Serial Self Refresh DRAM devices introduced by Cypress Semiconductor as a part of 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 cost. However, serial memories historically have lower data throughput or longer random-access time. This is mitigated in HyperBus and xSPI 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 as second-generation devices that now support a clock frequency of 200 MHz have a peak data transfer rate of 0.4 GB/s. Figure 1 provides a data throughput comparison across different interfaces.

Figure 1. Interface Data Throughputs



2 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 multi-layer 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.

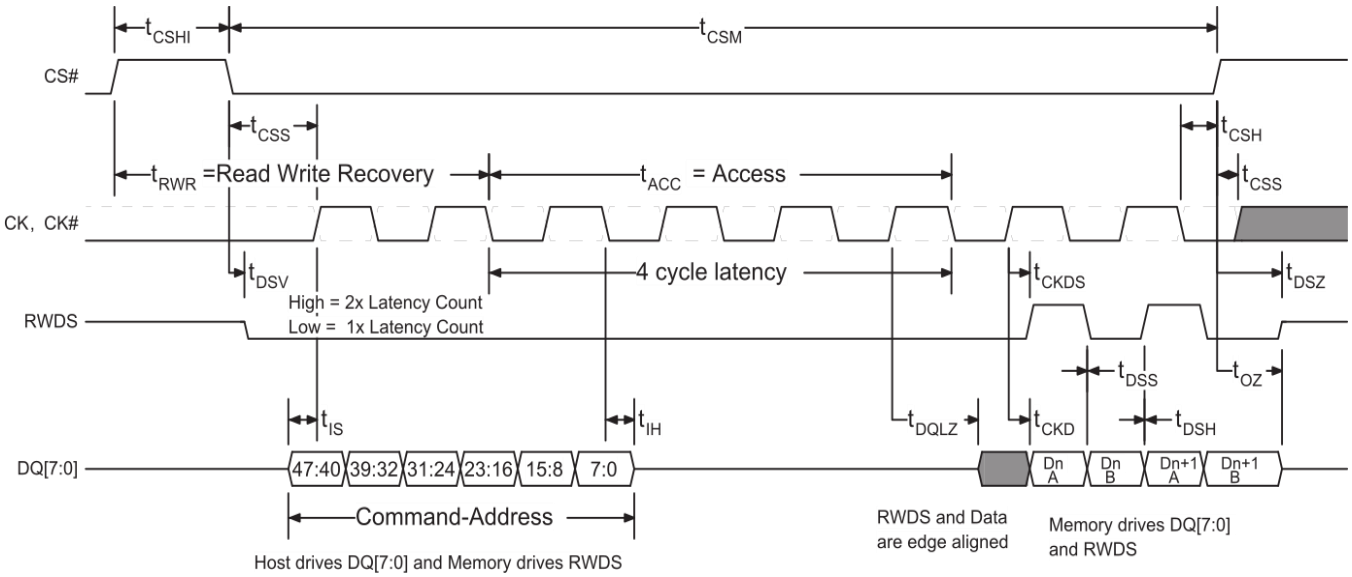
HyperBus has a low signal count, Double Data Rate (DDR) interface that achieves high read and write throughput while reducing the number of device I/O connections and signal routing congestion in a system.

These devices transmit data and command/address information in double data rate (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 read 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 a HyperBus Slave device when receiving 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 de-asserted.

Figure 2. Read Transaction, Single Initial Latency Count



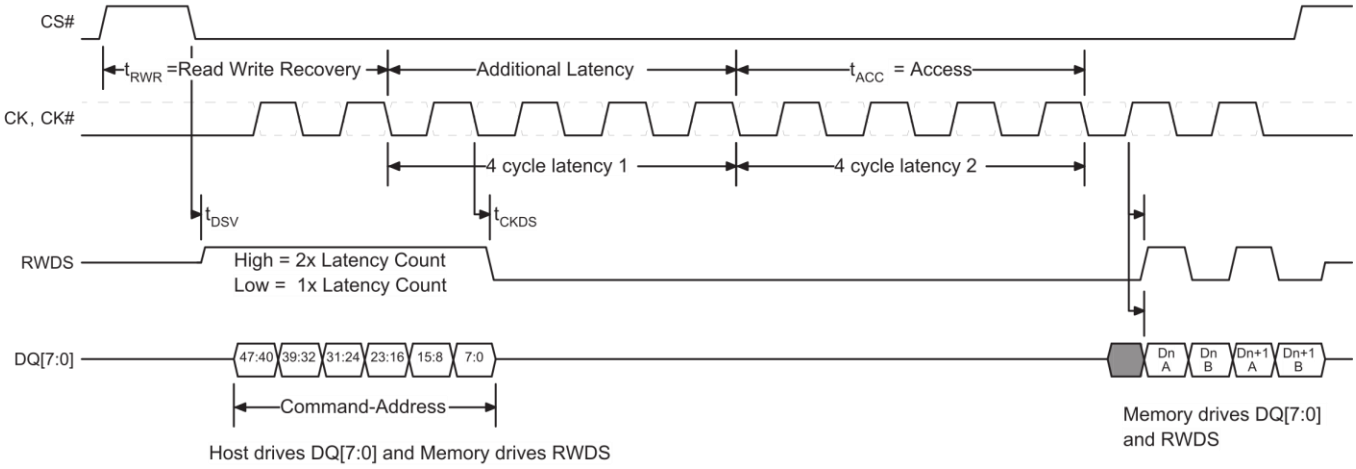
The Read/Write Data Strobe (RWDS) is a bidirectional signal that indicates:

- when data will start to transfer from a HyperRAM device to the Master device in read transactions (initial read latency)
- when data is being transferred from a HyperRAM device to the Master device during read transactions (as a source-synchronous read data strobe)
- when data may start to transfer from the Master device to a 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 a 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



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 a memory) or not masked with RWDS LOW (valid and written to a memory). Data masking may be used by the host to byte-align the write data within a memory or enable 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 detailed description of the HyperBus protocol and relevant timing parameters.

3 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 a wide adoption by microcontrollers and IP venders in market.

Each transaction on xSPI (Octal) must include a command whereas address and data are optional. The transactions are structures 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 (t_{RFH}) 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 that 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 (t_{RFH}), 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)

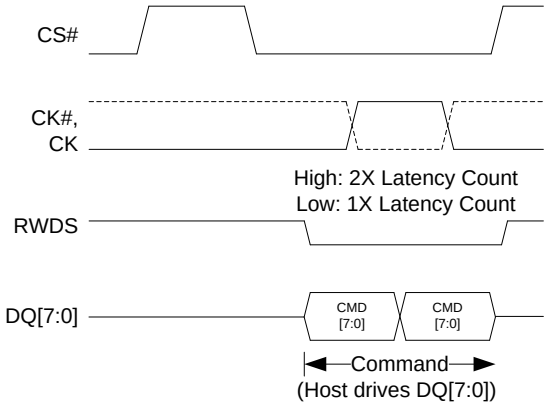


Figure 5. xSPI (Octal) Write with No Latency Transaction (DDR) (Register Writes)

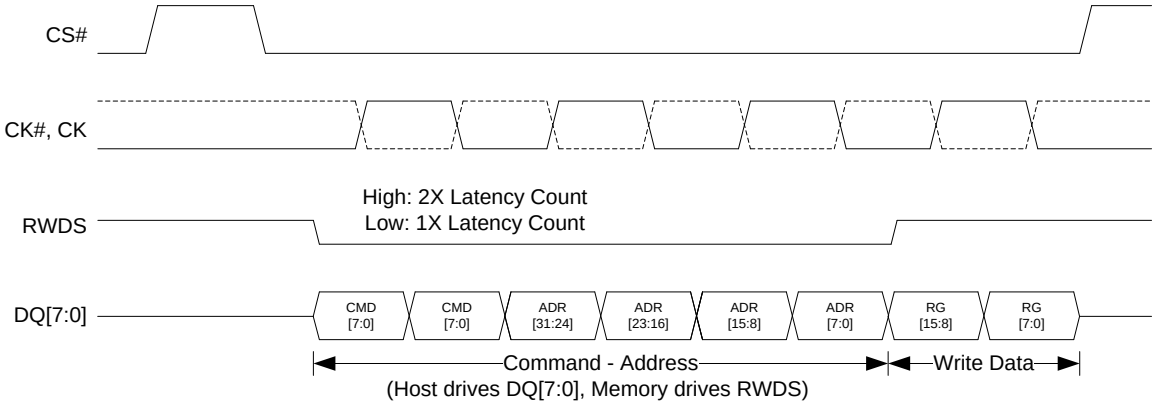


Figure 6. xSPI (Octal) Write with 1X Latency Transaction (DDR) (Memory Array Writes)

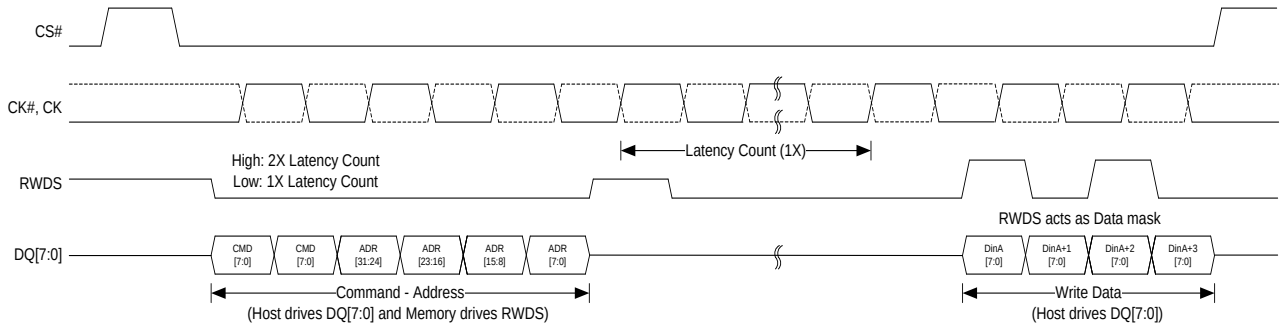
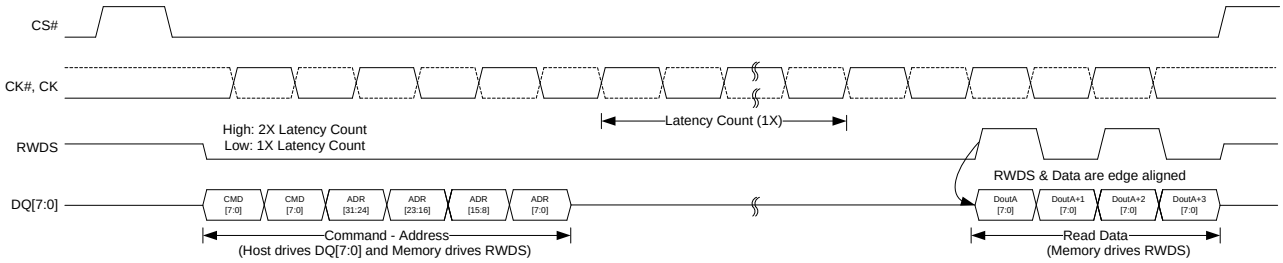


Figure 7. xSPI (Octal) Read with 1X Latency Transaction (DDR) (All Reads)



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

4 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 t_{ACC}. During the Command (CA) part of a transaction, the memory will indicate whether an additional latency for a required refresh time (t_{RFH}) 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 convention. Table 1 below summarizes all the signals of a HyperRAM device.

Table 1. Signal Description Summary

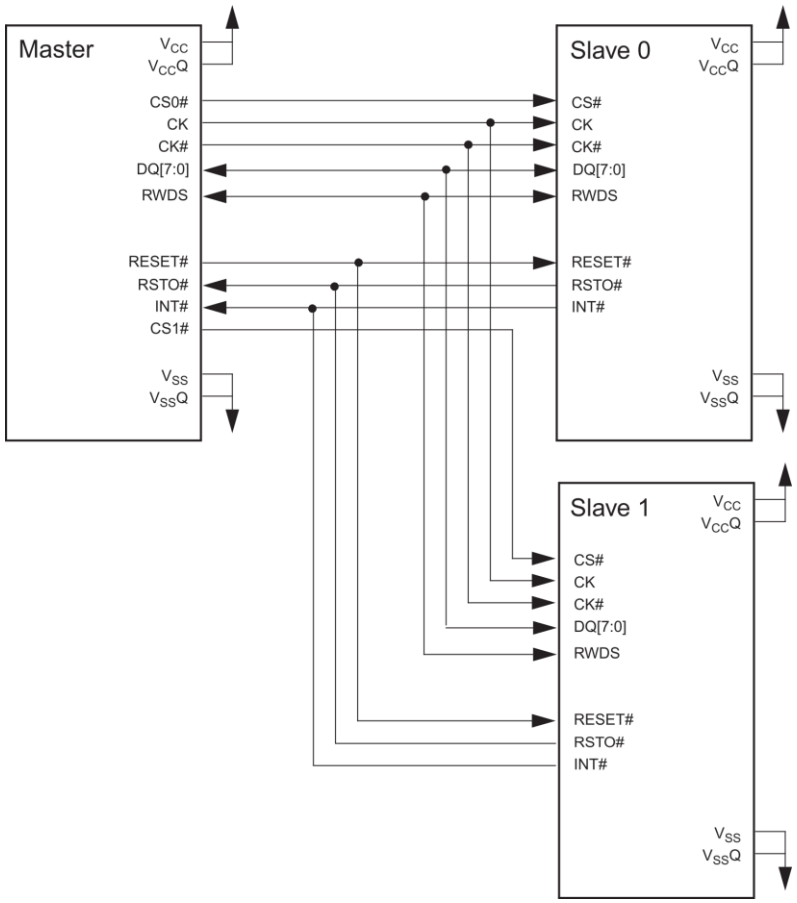
Pin Name	Type	Description
CS#	Input	Chip Select. Bus transactions are initiated with a HIGH-to-LOW transition. Bus transactions are terminated with a LOW-to-HIGH transition. The master device has a separate CS# for each Slave.
CK, CK#	Input	Differential Clock. Command, address, and data information is output with respect to the crossing of the CK and CK# signals. 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 Read 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 read 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 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.

Pin Name	Type	Description
VCC	Power Supply	Array Power.
VCCQ	Power Supply	Input / Output power.
GND VSS	Power Supply	Array Ground.
VSSQ	Power Supply	Input / Output Ground.
RFU	No Connect	Reserved for Future Use. May or may not be connected internally, the signal/ball location should be left unconnected and unused by PCB routing channel for future compatibility. 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 Slave can be a HyperFlash device while the other can be a 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 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 t_{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 read and write 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 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 write operation. The device automatically enables this state when clock remains stable for t_{ACC} + 30 ns. While in 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. Clock can be stopped during any portion of the active transaction as long as it is in LOW state. Note that it is recommended to avoid stopping the clock during register access. Refer to device datasheet for detailed specifications.

Deep Power Down: In Deep Power Down (DPD) state, current consumption is driven to the lowest possible level (i_{DPD}). 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 Standby state requires t_{EXTDPD} time. Returning to 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 following POR.

Note: In xSPI (Octal), Deep Power Down transaction or Write Any register transaction can be used to enter DPD.

Hybrid Sleep: This power is available only in 2nd generation of 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 HS state. Bringing CS# LOW will cause the device to exit HS state and relevant bit to '1'. Also, POR or a hardware reset will cause the device to exit Hybrid Sleep state.

Note that a POR or a hardware reset disables refresh where the memory core data can potentially get lost. Returning to 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 t_{RPH}, the device will draw I_{CC5} current. If RESET# continues to be held LOW beyond t_{RPH}, the device draws CMOS standby current (I_{CC4}). While RESET# is LOW (during t_{RP}), and during t_{RPH}, 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 Hybrid Sleep state
- Force the device to exit Deep Power Down state

After RESET# returns HIGH, the self-refresh operation will resume. Because 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 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 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.

5 Designing with HyperRAM

HyperBus and xSPI standards are widely getting adopted across industry as one of the leading low pin count, high throughput interface. Cypress Semiconductor works closely with HyperBus chipset parts to enable HyperBus interface as a hard IP in SoC or though Soft IP on leading FPGA vendors.

Table 3. HyperBus Chipset Support

Partner	Chipset / Platform Name	Application	HyperFlash	HyperRAM
Cypress	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 Series	Industrial	•	•
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, Industrial, Consumer	•	•
	i.MX RT1050	Industrial	•	•
	i.MX RT1020	Industrial	•	
	i.MX RT1060	Industrial	•	•
	i.MX RT106A	MCU-Based Solution for Alexa Voice Service	•	•
	I.MX RT family	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	Industrial	•	•
	Chorus SPC58 H	Automotive Gateway		•
Texas Instruments	Sitara AM6xxx	Industrial, Networking	•	•
Xilinx	VIRTEX Ultrascale+ (16nm)	Communications	•	•
	KINTEX Ultrascale+ (16nm)	Communications	•	•
	VIRTEX Ultrascale (20nm)	Communications	•	•
	KINTEX Ultrascale (20nm)	Communications / Industrial	•	•
	VIRTEX-7 (28nm)	Networking	•	•
	KINTEX-7 (28nm)	Networking	•	•
	Zynq 7000	Various	•	•
	Zynq UltraScale+	Industrial	•	•
	Artix-7	Industrial	•	•

Table 4. HyperBus 3rd Party Development Platform

Partner	3rd Party Development Platform	Application	HyperFlash	HyperRAM
Trenz Electronics	TE0725 with Xilinx Artix-7 TE0748 with Xilinx Artix	Industrial Secure SD	• •	• •
Devboards	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
Cypress	Link Cypress	•	•
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	•	•

Cypress Semiconductor also provides several design (IBIS and behavioral) models that allow customers to get a head start on system design.

6 HyperRAM – A Low-Pin-Count, High-Performance System Memory

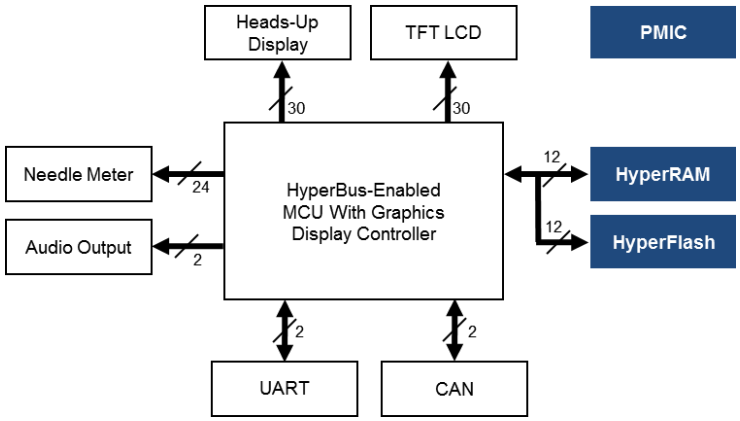
The size of internal RAM is one of the most important factors when determining how to execute complex algorithms with a controller or an FPGA/SoC. Adequate RAM or system memory allows code to be fetched and executed by the processor. This RAM may also double as Stack/Heap storage to be utilized by application 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 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.

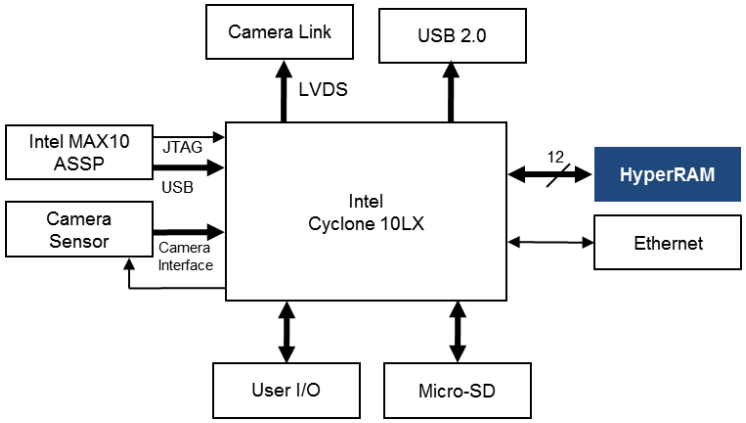
Figure 9 and Figure 10 show usage models of HyperRAM in systems. Automotive clusters perform blending of several graphics layer to generate the final display in a car. The quality and complexity of the graphics depends on size of the display buffer that can be implemented in the controller. Adding a high-performance HyperRAM to such systems enables loading and generating several complex display elements without having any penalty on 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 gets loaded from on-board flash or SD card onto the HyperRAM for highest performance.

Figure 10. HyperRAM as Expansion Memory in Machine Vision System



7 Related Documents

- AN211622 – HyperFlash and HyperRAM Layout Guide
- AN209853 – HyperRAM™ Refresh Interval Optimization
- AN218684 – HyperBus™ Memory: Guide to Efficient Data Access
- AN226137 – Migrating from S27KS0641 to S27KS0642
- HyperBus™ Specification Low Signal Count, High-Performance DDR Bus

About the Author and Contributors

Name:	Nilesh Badodekar
Title:	Sr. Staff Applications Engineer
Background:	Nilesh Badodekar has Master's degree in Visual Information Processing and Embedded Systems from Indian Institute of Technology, Kharagpur, India

Document History

Document Title: AN226576 – Getting Started with HyperRAM™

Document Number: 002-26576

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6591734	NILE	06/14/2019	New Application Note.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709