Introduction to HyperBus Memory Devices

Vignesh Raghavendra

Texas Instruments India

vigneshr@ti.com

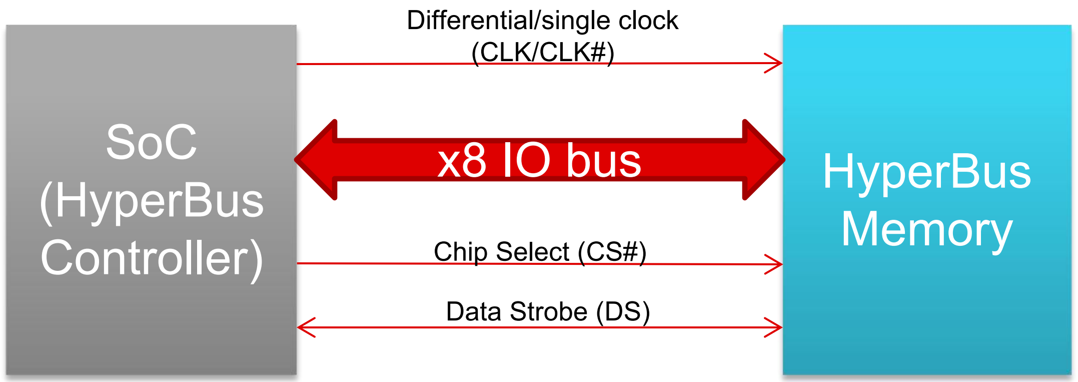
About me

- Software Engineer at Texas Instruments
- Co-maintainer of Memory Technology Devices (MTD) framework in kernel
- Presentation is based on learning that I had when adding HyperBus framework to support controller on TI’s AM654 SoC.

What’s in the presentation?

- HyperBus and types of HyperBus Memories
- HyperBus Protocol
- HyperFlash command set
- HyperBus Kernel Framework
- Writing a controller driver
- Recent developments and future enhancements

What’s HyperBus?


Differential/single clock (CLK/CLK#)
SoC (HyperBus Controller)
x8 IO bus
HyperBus Memory
Chip Select (CS#)
Data Strobe (DS)

- 8 data lines, Double Data Rate bus
- Single or Differential clocking
- Bi directional Data Strobe for accurate data capture

HyperBus Memory Devices

- On board embedded storage devices
- Two types of memory devices are available today
  - HyperFlash
    - Persistent Storage
  - HyperRAM
    - Pseudo static, volatile storage

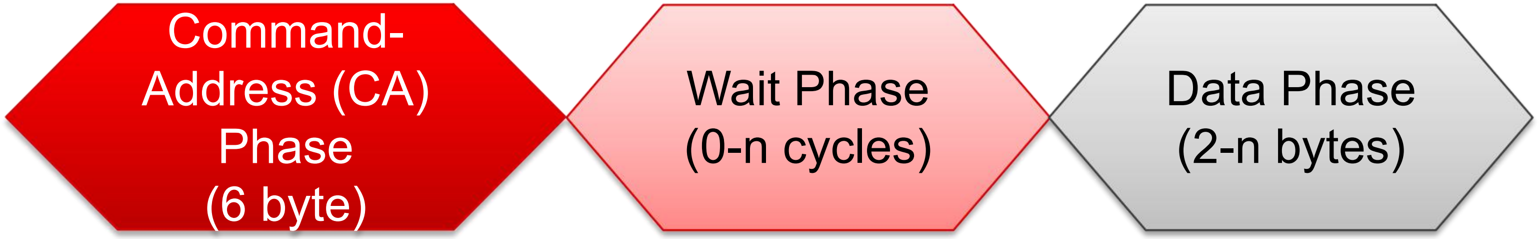
HyperFlash

- NOR technology based storage device
  - Organized into pages and sectors
  - 16 bit bus, 16 bit word size
- Unidirectional Data Strobe (Read Data Strobe)
- Can operate at up to 200MHz frequency
  - Read throughput can be as high as 400MB/s
- Draws upon the legacy features of both parallel and serial memories
- Alternative to Octal SPI NOR flashes

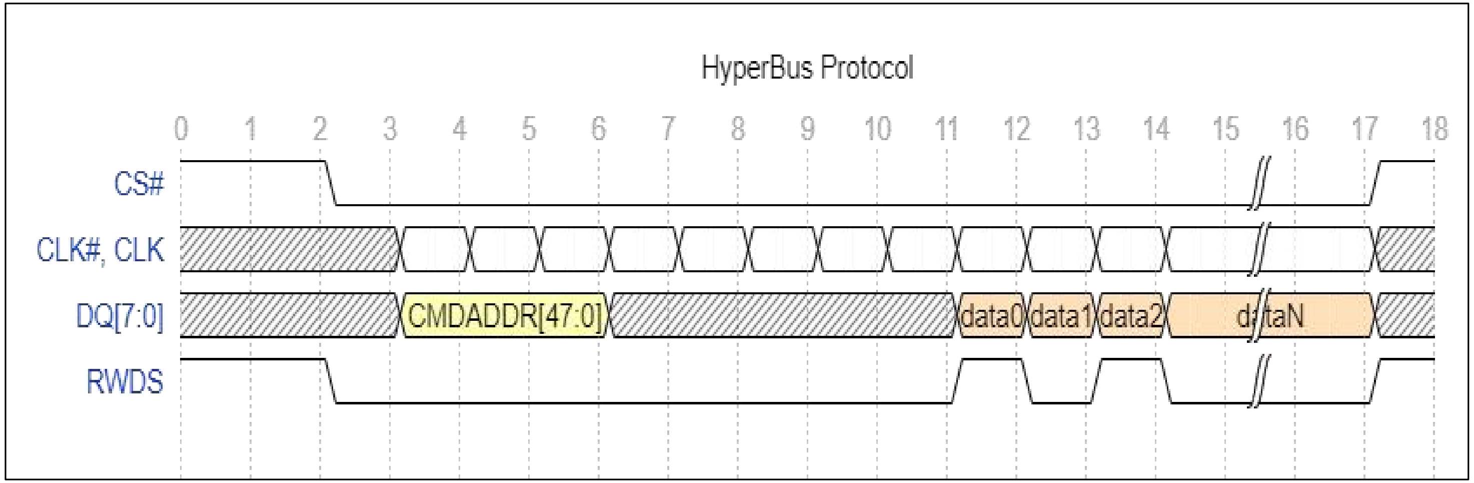
HyperRAM

- Self Refresh DRAM (Pseudo Static RAM) with HyperBus interface
- Same number of signals as HyperFlash
- Bi directional Data Strobe to indicate data validity and to handle additional latency during refresh

Phases of a transaction



Communication Protocol



Command Address (CA) Bits

Bit 47	Bit 46	Bit 45	Bits 44-16	Bits 15-3	Bits 2-0
R/W#	Target addr space	Burst type	Address (half-page selector)	Reserved	Address (word within half-page)
0: write 1: read	0: mem 1: reg	0: Wrapped 1: Linear	(A31- A3) 29 bits	Don’t care (Set to 0)	(A2-A0) 16 bytes

- Flash may be organized into half pages which is 16 bytes in size
- Half page refers to smallest region for which ECC is calculated

Programming sequence

- HyperFlash is compliant with Common Flash Interface (CFI) Extended Command Set 0002
  - Widely used by AMD/Fujitsu/Cypress Parallel NOR flashes
  - Driver implementing command set: drivers/mtd/chips/cfi_cmdset0002.c
- Flash powers up in read mode (default on POR or after HW or SW reset)
- Start read transaction(CA47 = 1) to desired address
  - Flash responds with data after predefined wait cycles
  - Needs to be 16 bit aligned addresses

Write Programming Sequence

- Sequence of writes to specific addresses make flash enter programming mode
  - Unlock1
    - Value: 0xAA → Address: 0x555
  - Unlock2
    - Value: 0x55 → Address: 0x2AA
  - PP command Write
    - Value: 0x25 → Address: Sector start address (SA) where buffer is present
  - Value: word count (no of bytes to update) → Address: SA
  - Value: data[0 – wc] → Address: start address (within Sector)
  - Value: 0x29 → Address: SA (confirm programming)
- Data is buffered before being written to flash
  - Supports Buffered writes of 512 bytes
- Note: Address refers to 16 bit word address

Address space overlays (ASOs)

- Different flash address spaces:
  - Flash memory array
    - Default region – where actual data is stored
  - ID/CFI space
    - Device ID and Common Flash Interface table
  - Status Registers
  - Persistent Protection bits
  - More vendor specific offerings
- ASO Entered/Exited by doing a specific sequences of writes
- Entire flash device address range or selected sector is overlayed with new region

Parallel CFI Flash vs HyperFlash

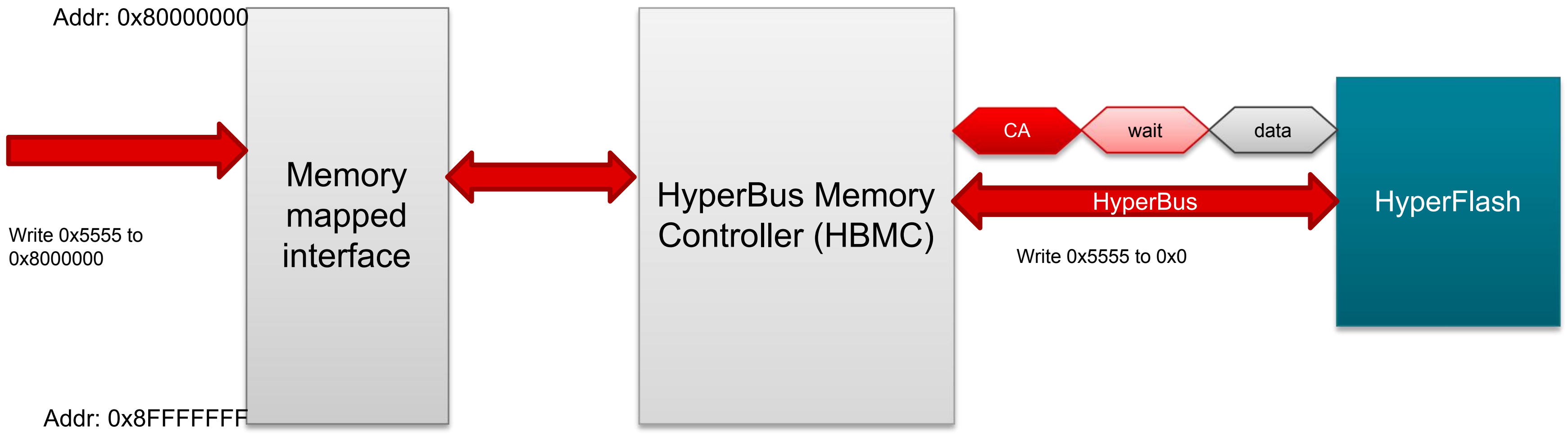
- Write/Erase completion polling
  - HyperFlash provides status register
  - Parallel NOR is mostly DQ polling
- Buswidth
  - Parallel NOR flash come in various buswidth and may be banked/interleaved
  - Single contiguous bank and 16 bit bus (x8 IO lines with DDR)
- Command set
  - Parallel NOR flashes have multiple different command set standards
  - Supports a single command set

Types of HyperBus Memory Controllers (HBMC)

- Dedicated HyperBus Controllers
  - Understands only HyperBus protocol
  - Support memory mapped IO (MMIO) access to flash
- Multi IO Serial controllers
  - Support multiple Serial Protocols such as: SPI NOR, SPI NAND, OSPI, HyperBus etc
  - May or may not support MMIO access to flash

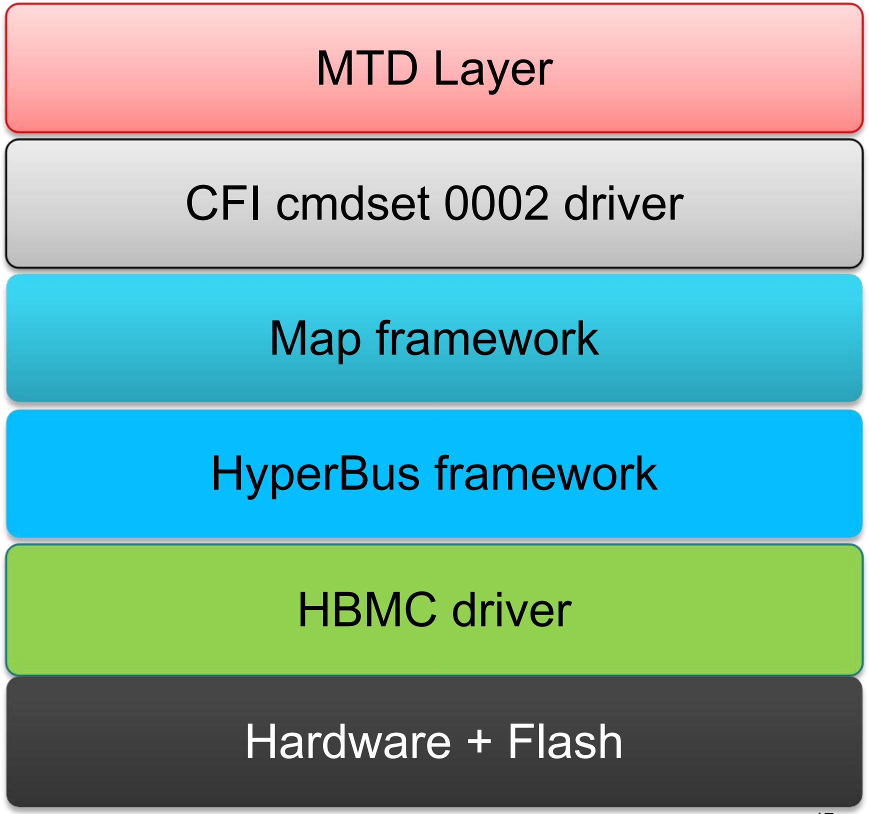
MMIO capable controllers

- Hardware can generates appropriate HyperBus transaction
  - Software need not manually provide 48 bits of CA phase
- Exposes entire flash to CPU at pre defined SoC Address space
  - Hence can also support XIP



Kernel support for HyperFlash

- HyperBus support merged in v5.3
  - Supports HyperFlash
  - Support MMIO capable HyperBus controller
  - Hooks upto existing CFI framework
- CFI layer implements command set
  - Legacy driver used with Parallel NOR flashes
  - Driver had to be modified to support Status Register polling to work with HyperFlash



Writing a HBMC driver

- Driver needs to implement **hyperbus_ops**

```
struct hyperbus_ops {
        u16 (*read16)(…)
        void (*write16)(…);
        void (*copy_from)(…);
        void (*copy_to)(…);
        int (*calibrate)(…)
};
```

hyperbus_ops

- **read16()**
  - Read 16 bit of data from flash in a single burst.
  - Used to read from non default address space, such as ID/CFI space
- **write16()**
  - Write 16 bit of data to flash in a single burst.
  - Used for non default address spaces as well as for single word programming
- **copy_from()**
  - Read data from flash memory array
- **copy_to()**
  - Write data to flash memory array
- **calibrate**()
  - Calibrate controller by using a known data pattern

Registering Device

- Populate per device struct:

```
struct hyperbus_device {
        struct map_info map;
        struct device_node *np;
        struct mtd_info *mtd;
        struct hyperbus_ctlr *ctlr;
        enum hyperbus_memtype memtype;
};
```

- Register each driver with core:

```
int hyperbus_register_device(struct hyperbus_device *hbdev);
```

Device Tree representation

```
hbmc: memory-controller@47034000 {
                compatible = "ti,am654-hbmc";
                reg = <0x0 0x47034000 0x0 0x100>,
                        <0x5 0x00000000 0x1 0x0000000>;
                #address-cells = <2>;
                #size-cells = <1>;
                ranges =        <0x0 0x0 0x5 0x00000000 0x4000000>, /* CS0 - 64MB */
                                <0x1 0x0 0x5 0x04000000 0x4000000>; /* CS1 - 64MB */

                /* Slave flash node */
                flash@0,0 {
                                compatible = "cypress,hyperflash", "cfi-flash";
                                reg = <0x0 0x0 0x4000000>;
                };
};
```

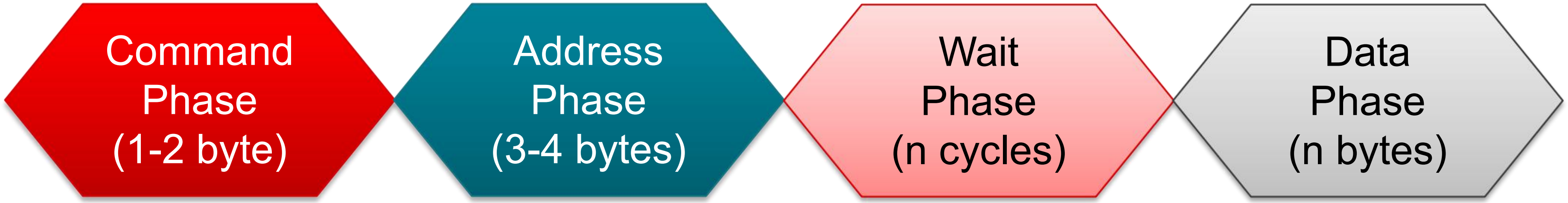
Accessing from user space

- Just like any other MTD devices
  - Device is exposed as /dev/mtdX to user space
- Use mtd-utils
  - http://git.infradead.org/mtd-utils.git
- Flash Filesystems such as UBIFS can be used.

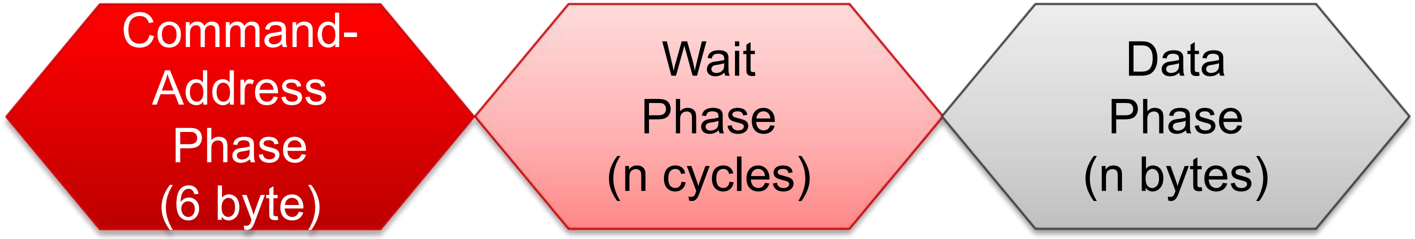
HyperFlash and xSPI

- HyperFlash protocol is now part of JEDEC xSPI specification
  - JESD 251: Extended SPI specification
- xSPI aims to standardize command set and protocol for programming of Serial flashes
- HyperFlash protocol is described in Profile 2.0 of xSPI specification
  - Separate profile than traditional SPI NOR

Comparison to traditional SPI flash protocol



SPI flash protocol



HyperBus protocol

xSPI compliant HyperFlash

- HyperFlash powers up in SPI mode (1S-1S-1S)
  - Backward compatible with legacy SPI commands
    - Transaction phase: 1 byte cmd – 3 byte address - data
- Can be configured to work in HyperBus mode (CA-data) mode by setting a configuration register bit
- Serial Flash Discoverable Protocol (SFDP) table: JESD216D
  - Discover and configure flash in manufacturer agnostic way

Extending spi-mem for HyperFlash

- SPI subsystem has spi-mem layer that abstracts SPI memory devices
- spi_mem_op currently supports cmd-addr-data
  - cmd is 1 byte
  - upto 4 byte addressing
- Supporting dual protocol capable flashes (SPI and HyperBus) and such controllers would need update to spi-mem-op template
  - Add new member to indicate HyperFlash mode
  - Extend cmd and address field to accommodate HyperFlash protocol
- HyperBus and SPI NOR core drivers can then use spi_mem_ops to support dual protocol capable flashes

Future Enhancements

- Writes are done at word granularity
  - Can be improved to work at buffer granularity
- DMA support for reading data from flash
  - Handling vmalloc'd buffers passed from Flash filesystems like UBIFS
- Use spi_mem_ops
  - Support Multi protocol SPI controllers
  - Support xSPI complaint SPI and HyperBus compatible devices.

References

- Specification and flash datasheets
  - HyperBus specification: https://www.cypress.com/file/213356/download
  - HyperFlash: https://www.cypress.com/file/213346/download
  - HyperRAM: https://www.cypress.com/file/183506/download
- Source:
  - https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/mtd/hyperbus

Credits

Thanks

- Texas Instruments Inc.
- The Linux Foundation

Q & A

E-mail: vigneshr@ti.com

Thank You!