Freescale Semiconductor, Inc.
Application Note

Document Number: AN4947
Rev. 0, 07/2014

Understanding Vybrid Architecture

by *Jiri Kotzian and Rastislav Pavlanin*

Vybrid controller solutions are built on new asymmetrical multiprocessing architecture using ARM® cores. The purpose of this application note is to provide some details of the Vybrid controller solutions system architecture. How an application is programmed into the Vybrid system-on-chip (SoC) has a significant impact on application performance, for instance, in how the application is divided and spread between cores, how peripherals are set, and which memories and caches are used. This document is focused on latencies of memories and cache setting. Several tests are included and results are presented in tables.

Contents

1 Reducing latency

Powerful cores and coprocessors are required for computing and processing large amounts of data, and high throughput is mandatory for multimedia applications, but for real-time control, latency is the most important factor. Latency is the time delay between cause and effect — for example, the time between a data request and the receipt of data. Vybrid solutions are intended for rich applications in real-time and, to achieve the goal of limited latency, the application must be programmed in a way that takes into account its architectural features and restrictions.

2 Vybrid architecture

Vybrid controller solutions include one Cortex®-A5 core for the rich part of the application and one Cortex-M4 for the real-time control part of the application, integrated with a large set of peripherals interconnected by the Network Interconnect (NIC), which is the key point of the architecture.

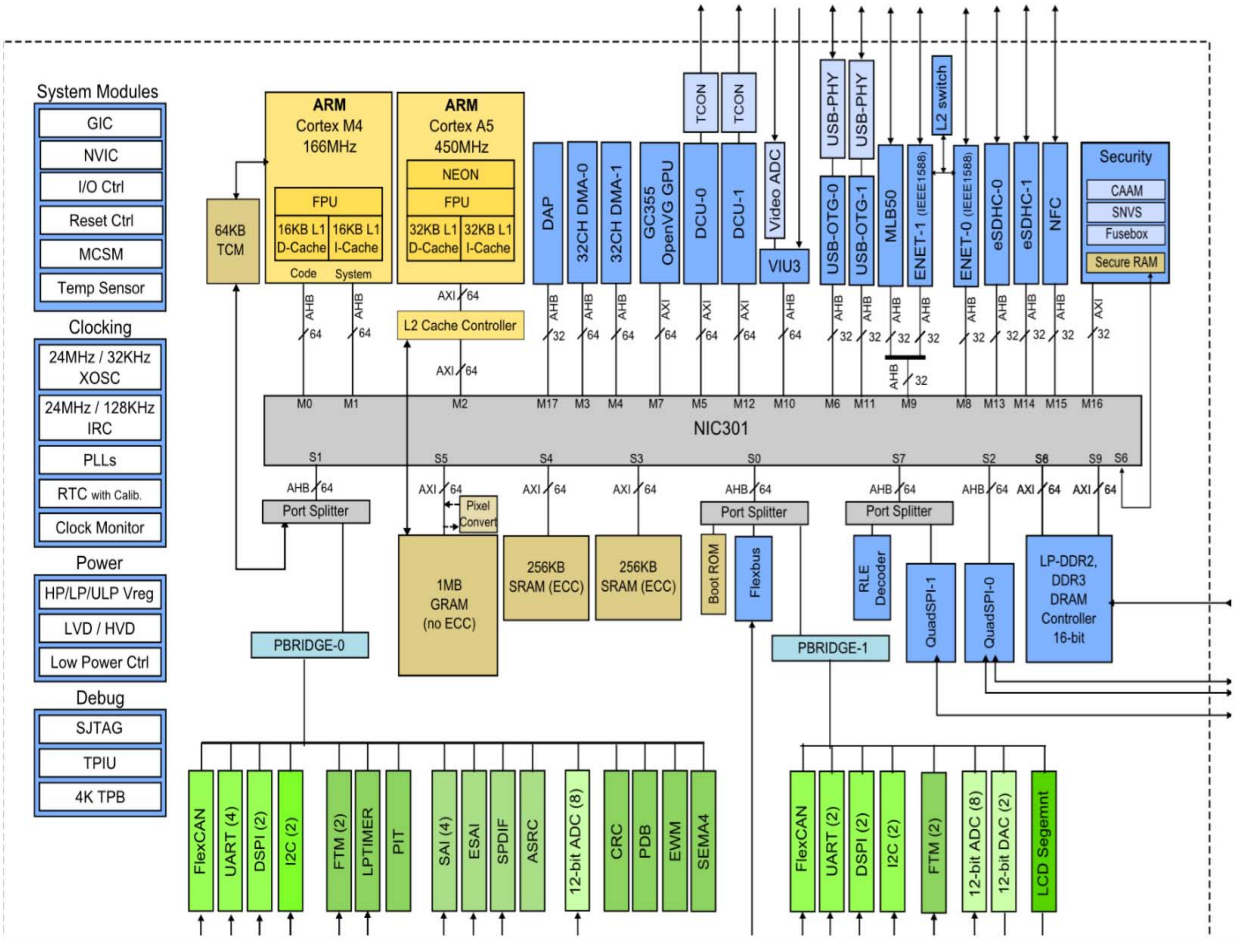

Figure 1. Vybrid internal architecture

Figure 1 shows the internal architecture of the Vybrid controller solutions. Vybrid controller solutions integrate the following parts:

- ARM Cortex-A5: Includes FPU single and double precision, NEON Media Processing Engine, ARM or thumb mode (32 or 16-bit instructions), 1.57 DMIPS per MHz integer performance, eight-stage single issue pipeline, four-stage load/store pipeline, and five-stage FPU/MPE pipeline, up to 500 MHz.
- ARM Cortex-M4: Includes single-precision FPU, DSP and SIMD ISA extensions, thumb2 mode only (16-bit instructions for smaller code size), three-stage pipeline, and 1.25 DMIPS per MHz performance, up to 166 MHz.
- NIC-301: High-performance optimized AMBA-compliant network infrastructure providing hardware interconnect matrix between 10 masters and eight slaves. The NIC is optimized for 64-bit AXI interface. NIC system bus fabric optimized for multi-master data bandwidth.

- Internal memories TCM, SRAM, Secure RAM, and GRAM
- External memory controllers for SDRAM, QuadSPIs, FlexBus and SDHC
- DMA, semaphores, security, TrustZone
- Set of peripherals including Ethernet, USB, CAN, SCI, SPI, I^2C, audio and display drivers.

For more information, see the *Vybrid Reference Manual* [2] and the *Vybrid Data Sheet* [3].

3 Architectural key points

How an application is programmed into the Vybrid SoC has a significant impact on application performance — specifically, how the application is divided and spread between cores, how peripherals are set, and which memories and caches are used.

3.1 Clocks

Vybrid controller solutions support several clock domains:

- Cortex-A5 clock; 266, 400–500 MHz
- Platform clock (NIC, Cortex-M4 clock); 133–166 MHz; usual ratio 1:3 to Cortex-A5 clock
- DDR clock; max 400 MHz
- Peripheral; max 83 MHz; usual ratio 1:2 to platform clock.
- Etc.

Find more information in *Vybrid Reference Manual* [2], chapter “Clock Configuration.”

3.2 NIC-301

The most important part of the architecture is the NIC-301, displayed in the center of Figure 1. The NIC creates interconnections between masters (cores and DMAs) and slaves (peripherals and memories). All data transfers go through these interconnections. The NIC architecture determines maximum throughput between masters and slave, but also adds latency to transfers between them.

3.2.1 Sharing NIC nodes

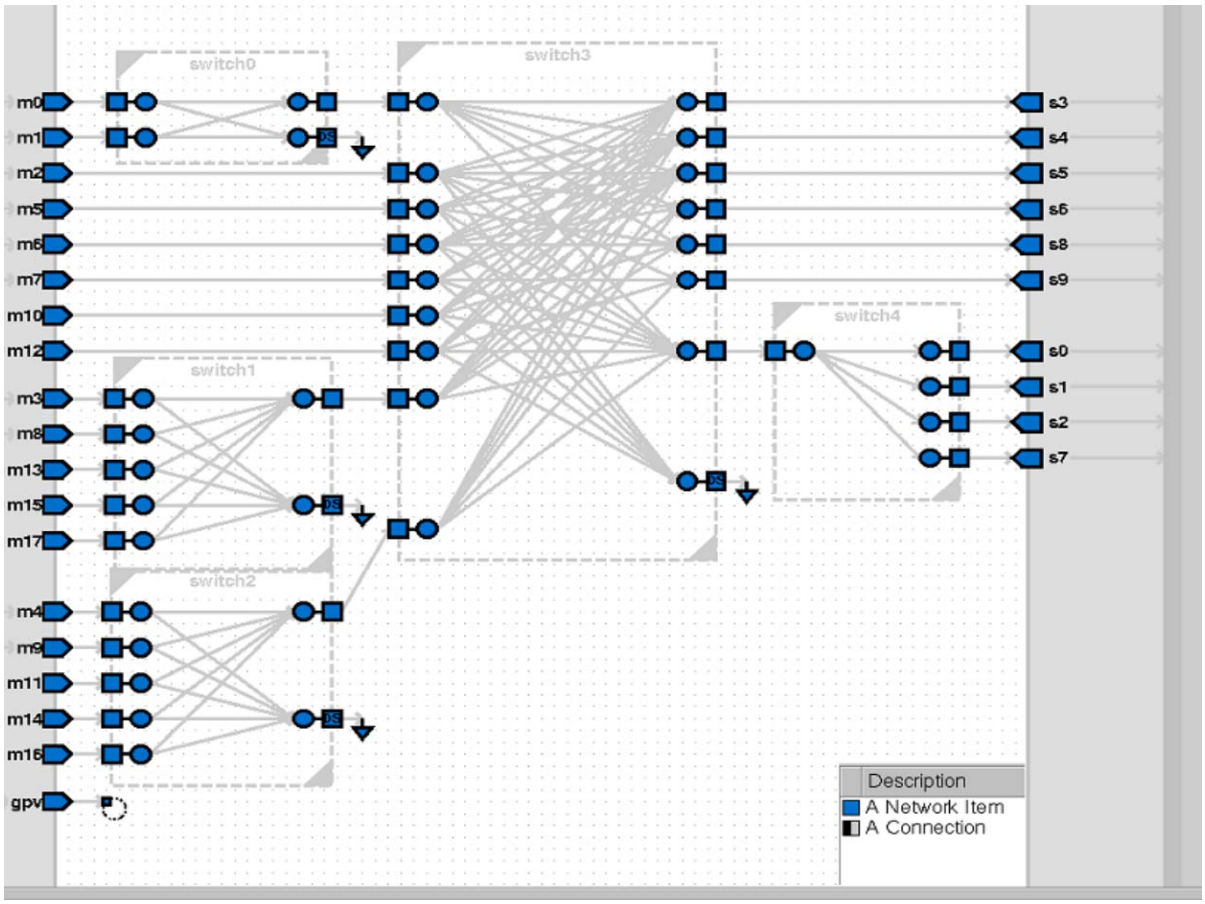

Figure 2. NIC internal structure

Figure 2 shows the internal structure of the NIC. The Vybrid platform is too large to be implemented by a single monolithic switch. It is actually implemented as five sub-switches which act as one logical switch fabric (sub-switches 0–4 under "switch" heading). However, NIC is able to connect 18 masters and 10 slaves with each other using master nodes M0 to M17 and slave nodes S0 to S7.

The limitations on NIC are:

- Secure RAM (NIC S6) cannot be accessed by DCU, VIU2 and (OpenVG).
- SDRAM port (NIC S8) can be accessed by Cortex-M4, DMA0, DCU0, USB0, ENET0, VIU2, SDHC0, NAND, CA5 DAP only.
- SDRAM port (NIC S9) can be accessed by Cortex-A5, DMA1, DCU1, USB1, ENET1, (OpenVG), SDHC1 and security only.
- Each switch input can be connected to only one output (and vice-versa).

For example. the Cortex-A5 core which is the master (NIC M2 node) on the NIC wants to communicate with SRAM memory on node S4. In this case the internal switch, *switch3*, connects the second input with the second output (nodes M2-S4). At the same time, no other master can communicate with the memory. If a different master with higher priority needs to communicate, for example DMA0 M4, the connection

is interrupted and M4 is connected with S4 using internal switches *switch2* and *switch3*. As can be expected, the transition to a different transfer takes some time and creates additional latency.

To avoid frequent switching we must carefully design our application. For example, the code and data of each core can be stored in one SRAM memory bank (NIC S4 node). In this case, both cores will fight for the memory and *switch3* will switch frequently between *switch0* output and M4 node.

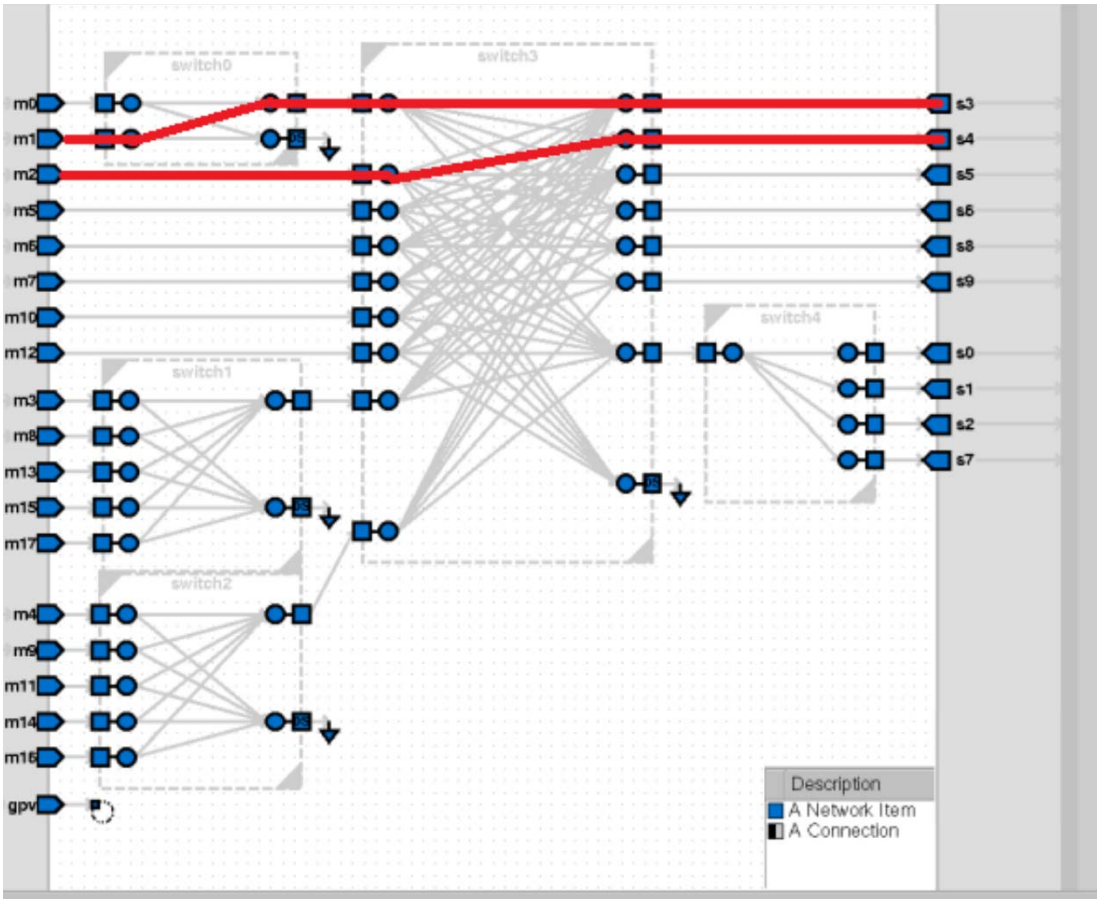

Figure 3. NIC data transfers without additional latencies

A better solution is to divide code and data between separate memory banks on nodes S3 and S4, shown in Figure 3. In that case *switch3* will be set to one state and transfers can occur without additional latency. Cortex-M4 core on M0/M1 node will communicate with SRAM memory bank 1 on S3 node and Cortex-A5 core on M2 node will communicate with SRAM memory bank 2 on S4 node without additional latencies.

3.3 NIC priorities

Each master node on the NIC has an assigned priority. Priorities can be changed by the PBRIDGE0 bus controller, but this is not recommended. Default priorities can be found in the *Vybrid Reference Manual* [2] (NIC301 Default QoS Arbitration Priorities).

3.4 NIC transfer latency

The NIC provides a complete crossbar connection between masters and slaves. The NIC is also highly configurable, supporting a variety of bus widths, bus protocols, clocking, and both timing and architectural buffering.

Each communication on the NIC takes multiple platform clocks depending on: which master is selected, which slave is selected, buffering method, and type of transfer. Data are presented in Table 1 and Table 2.

Table 1. NIC latencies on master nodes

NIC Master	Description	Data width	Protocol	Read Issue	Write Issue
M0	Cortex-M4-PC	64	AHB	2	4
M1	Cortex-M4-PS	64	AHB	2	4
M2	Cortex-A5	64	AXI	8	12
M3	32-eDMA0	64	AHB	2	4
M4	32-eDMA1	64	AHB	2	4
M5	DCU0	64	AXI	6	1
M6	USB0	32	AHB	2	4
M7	Open VG	64	AXI	4	4
M8	ENET0	32	AHB	2	4
M9	ENET1	32	AHB	2	4
M10	VIU2	64	AHB	2	4
M11	USB1	32	AHB	2	4
M12	DCU1	64	AXI	6	1
M13	eSDHC0	32	AHB	2	4
M14	eSDHC1	32	AHB	2	4
M15	NAND FLASH	32	AHB	2	4
M16	Security	32	AXI	2	4
M17	CA5 DAP	32	AHB	2	4

Table 2. NIC latencies on slave nodes

NIC Slave	Description	Data width	Protocol	Read Accept	Write Accept
S0	32K ROM	64	AHB	1	1
S1	CM4-TCML	64	AHB	1	1
S2	QuadSPI0	64	AHB	1	1
S0	FlexBus	32	AHB	-	-
S3	SRAM1	64	AXI	4	4

Table 2. NIC latencies on slave nodes (continued)

NIC Slave	Description	Data width	Protocol	Read Accept	Write Accept
S4	SRAM2	64	AXI	4	4
S5	GRAM	64	AXI	4	4
S1	CM4_TCMU	64	AHB	-	-
S0	AIPS0	32	AHB	-	-
S6	SecureRAM	32	AXI	2	2
S1	AIPS1	64	AHB	-	-
S7	QuadSPI1	64	AHB	1	1
S7	RLE	64	AHB	-	-
S7	QS1 RxBuf	64	AHB	-	-
S2	QS0 RxBuf	64	AHB	-	-
S5	GRAM	64	AXI	-	-
S8	SDRAMC	64	AXI	8	8
S9	SDRAMC	64	AXI	8	8

Latencies for read/write are in the *Read Issue/Write Issue* column for a given master, etc. Every transfer has to be accepted by the selected slave. This will bring another latency, as shown in the *Read Accept/Write Accept* column.

3.5 NIC buffering

The buffering is done via a register or FIFO, which adds a cycle of latency. The NIC can build a switch without any buffering providing there are no bus protocol conversions, no data width conversions, and no timing manipulations (n to 1, 1 to m, n to m clock ratios or async operation). That is, if every connection, both master and slave, were AXI bus protocol buses of the same frequency and data path width, the switch could be realized without any buffering. The basic structure of the Vybrid AXI switch is AXI protocol at platform frequency with 64-bit read/write data path. Buffering is forced by:

- Clock changes on all channels at the master or slave port — for example, the Cortex-A5 M2 port — core/platform clock change.
- Data width changes on at least some channels at the master or slave port — for example, the eSDHC0 M13 port — 64-bit/32-bit change.
- Protocol changes on at least some channels at the master or slave port — for example, ROM's S0 port — AXI to AHB protocol conversion.

On the master side, the buffering latency for read is usually zero to two clock cycles and the buffering latency for write is usually four clock cycles, except for DCU, which is zero.

On the slave side, buffering latency for read is usually two clock cycles and the buffering latency for write is usually four clock cycles.

3.6 Bus width and the transfer through NIC

The speed of the transfer through the NIC depends on who is asking. The NIC generally is able to communicate between masters and slaves in 32- to 128-bit bus wide. Vybrid uses 32- or 64-bit bus width, depending on given master/slave bus width.

The initial phase of burst transfer depends on traffic on the NIC, necessary buffering and master's priority (10 initial clocks should be the worst case scenario).

If given master supports burst transfer, the communication length depends on source/destination transfer size. See Figure 1 for bus widths and the *Vybrid Reference Manual* [2] for data transfer sizes.

Examples:

1. **The core to the ROM memory without the cache usage**: A core will communicate through 32 bits only (bus width). Every communication takes minimally three clock cycles (two NIC master, one NIC slave) + buffering latency.
2. **The core to the ROM with the cache usage:** The cache controller is between core and memory. Cache controller interface to the NIC is 64b (like the ROM memory). It is two times faster compared to previous examples due to 32b to 64b transition. The cache line sizes are 32B. This invokes burst mode on the NIC. The first read will be minimally three clock cycles (two NIC master, one NIC slave) latency; any others will be one additional platform clock cycle. Vybrid + cache uses 64 bits — in every clock 64b/8 = 8B are transferred. 32B line size loaded four clock cycles + start (two NIC master, one NIC slave) + buffering latency.
3. **DMA.** DMA supports 8b, 16b, 32b and 32B transfer size, and 64b interface. In the case of 32B transfer size, DMA transfer will take minimally four clock cycles + start three clock cycles (two NIC master, one NIC slave) + buffering latency.

3.7 Locking NIC

Transfer connection to SRAMs, SDRAM, GRAM and SecureRAM can be locked to Cortex-A5 or Cortex-M4 to prevent additional latency. Consequently, it disables other slaves to communicate with given memory. For more information, see the *NIC-301 Technical Reference Manual* [4].

3.8 Cache usage

Caches can significantly improve performance of the application and reduce on-chip memory. In the case of usage cache, memory data can be stored in the cache. The first time data are needed, it must be fetched from the memory and transferred through the NIC with additional latency. When used a second time, data are loaded from the cache in one clock cycle (L1 cache).

CM4 caches:

- Two-way set-associative 16 kB L1 instruction cache with 32B line size
- Four-way set-associative 16 kB L1 data cache with 32B line size

Cortex-M4 cache is controlled by the local memory controller — see the *Vybrid Reference Manual* [2] for more information.

CA5 caches:

- Two-way set-associative 32 kB L1 instruction cache with 32B line size
- Four-way set-associative 32 kB L1 data cache with 32B line size
- Eight-way set-associative 512 kB L2 cache with 32B line size

CA5 does not have an internal L2 cache controller. The external ARM L2C-310 is integrated on the Vybrid platform. L2 cache controller optionally uses the upper half of GRAM memory. GRAM memory runs on platform clock, while the controller itself runs on the Cortex-A5 clock. This limits throughput of L2 cache.

The external cache controller is connected between Cortex-A5 and the NIC. Therefore, even when the L2 cache controller is disabled, there are always four clocks of L2 cache controller penalty for all passed data. The same goes for L2 cache miss.

Table 3 shows typical memory sizes and access times for Cortex-A processors.

Table 3. Typical memory sizes and access times (source: Cortex-A Series Programmer's Guide) [6]

Memory type	Typical size	Typical access time
Processor registers	128B	1 cycle
On-chip L1 cache	32kB	1-2 cycles
On-chip L2 cache	256kB	8 cycles
Main memory, L3, dynamic RAM	MB or GB	30-100 cycles
Back-up memory, hard disk, L4	MB or GB	> 500 cycles

3.9 Cache direct access and lockdown

A cache preload is a powerful way to improve real-time performance of the application. Despite longer startup time of the application, the first run is as fast as any other run. The Vybrid platform offers different options depending on the core:

- **CM4:** The L1 cache tag and data storage arrays can be read and written indirectly via the cache controller's programmer's model. Using this, it is possible to manually dump or load the caches. There is no cache lock function. For usage cache at extended TCM without additional latency, we need to ensure that data size (code/system) is less that code/system cache size. There are predefined regions which are cached and those which are not cached. It is not possible to select it individually. You can turn on/off the code/system cache. The next option is to force all cacheable regions to write through or force no allocate, but they are dedicated for debug purposes.
- **Cortex-A5 L1 cache**: The Cortex-A5 provides a mechanism to read the internal memory used by the cache and TLB structures through the implementation-defined region of the system coprocessor interface. This functionality can be useful when investigating issues where the coherency between the data in the cache and data in system memory is broken (*ARM Cortex-A5 MPCore Technical Reference Manual*) [9]. Lockdown for instruction cache controller is not supported.
- **Cortex-A5 L2 cache**: There is no way to indirectly read or write the L2 cache. You can lock the replacement algorithm on a way basis, enabling the associativity to be reduced from 16-way down

to one-way (*L2C-310 Technical Reference Manual*). Vybrid L2 is configured as eight-way associative.

3.10 Cache miss and caches cooperation

Not all data can be stored in the cache. If data are not in the cache, data have to be fetched from source memory — this is called cache miss.

For **Cortex-M4** if **L1 cache miss** occurs, **one clock cycle** latency is taken and then we have to go through NIC to given memory. This causes additional latency, described in the "NIC Transfer Latency" chapter of *Cortex-A Series Programmer's Guide* [6]. When L1 cache misses occur, then 32B are loaded in burst mode so following opcodes are delayed less.

For **Cortex-A5** if **L1 cache miss** occurs, **one clock cycle** latency is taken and then we proceed to L2 cache if enabled, or we have to go through NIC to given memory. In the case that L2 cache is enabled, cache access is done in two phases:

1. **During the first phase** of L2 access, the tags are accessed in parallel and hit/miss is determined. The tag arrays are built in the L2 cache controller. No wait states are necessary.
2. **During the second phase** of a L2 access, a single data array structure is accessed:
   a) **L2 read cache hit**: the L2 data array provides one line of data to the L2 cache controller at the same time. For Vybrid, it takes three clock cycles to read or write the L2 data array structure (upper GRAM on platform frequency). Together, **four clocks Cortex-A5** latency.
   b) **L1 and L2 miss**: the L2 adds two clock cycles on the miss request and two cycles on the miss return. This four-cycle penalty is present even if the L2 is not enabled. Moreover, data have to be fetched from source memory. Together latency is **4 + 3x (NIC latency + source memory) Cortex-A5 clocks**.

One important element to consider when dealing with a cache miss is the miss rate. A cache miss rate depends on the application, especially on features like branch prediction, micro TLB hit/miss, instruction/data pipeline interaction and code character repeatable/linear. Some raw data for estimation indicates:

- Real data from different test ARM mode 4%, THUMB 2%
- Designer's estimate of worst case and linear code 20%, normally 2%

For more information about cache function, see *Cortex-A Series Programmer's Guide* [6], chapter "Caches."

3.11 Selecting the right memory

Vybrid integrates several memories which can be used by Cortex-A5 or Cortex-M4:

- SRAM 2x256 kB (Cortex-A5, Cortex-M4)
- GRAM 1 MB (Cortex-A5, Cortex-M4)
  — L2 cache shares 512 KB from GRAM (Cortex-A5)
- TCM (TCL 32 kB code bus, TCU 32 kB system bus), (Cortex-A5 by back door, Cortex-M4 directly)

We can also use external memories with integrated memory controllers for SDRAM (LPDDR2 and DDR3), QuadSPIs, FlexBus, NAND and SDHC.

3.11.1 Cortex-M4 tightly coupled memory (TCM)

The TCM is standard SRAM memory which is connected directly to Cortex-M4 via local memory controller (see Figure 4). It is divided into two parts: the lower half connected on code bus and the upper half connected on system bus. Therefore, Cortex-M4 can access TCM within one clock cycle — without additional latency. Accordingly, it is recommended to use TCM for code/data that need to be accessed frequently and/or with minimal and stable latency. The best example is ISR code and process data variables. TCM can be accessed by Cortex-A5 core using the back door. This access is slower because it has to go through NIC and port splitter (see Figure 1). The back door is intended to load the code and data into TCM by Cortex-A5 (primary core) prior to running Cortex-M4 (secondary core). Using TCM by Cortex-M4 core allows true real-time performance of the application to be achieved.

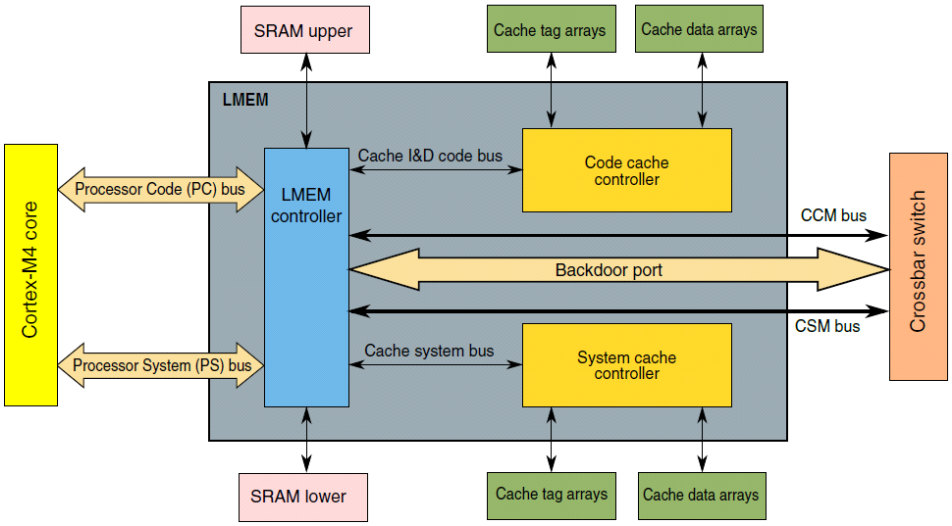

Figure 4. TCM and cache connected to Cortex-M4

3.11.2 Cortex-M4 memory aliases

Cortex-M4 uses modified Harvard architecture with code and system buses, where the address defines which bus is used. Instruction and vector fetches on the system bus (addr >= 0x2000_0000) are registered internally with an additional cycle of latency.

Table 4. Vybrid memory aliases for Cortex-M4

CM4 Address Range [Start Addr – End Addr]	Size (MB)	CM4 Alias	System Address (A5) [Start Addr – End Addr]	Region Description
0x0000_0000-0x007F_FFFF	8	0x0000_0000	0x0000_0000-0x007F_FFFF	Boot ROM
0x0080_0000-0x0FFF_FFF	248	0x8080_0000	Reserved	CM4 DDR code alias
0x1000_0000-0x17FF_FFFF	128	0x2000_0000	Reserved	CM4 QuadSPI0 code alias

Table 4. Vybrid memory aliases for Cortex-M4 (continued)

CM4 Address Range [Start Addr – End Addr]	Size (MB)	CM4 Alias	System Address (A5) [Start Addr – End Addr]	Region Description
0x1800_0000-0x1EFF_FFFF	112	0x3000_0000	Reserved	CM4 FlexBus code alias
0x1F00_0000-0x1F7F_FFFF	8	0x3f00_0000	Reserved	CM4 OCRAM code alias
0x1F80_0000-0xFFF_FFFF	8	N/A	0x1f80_0000-0x1FFF_FFFF	CM4 TCML (code)

Memory aliases were defined to cancel additional latency in the Cortex-M4 core — see Table 4. As a result, Cortex-M4 thinks that it is accessing memory under 0x2000_0000, so it is not adding one clock cycle, but it physically accesses memory above 0x2000_0000, to where the alias points.

3.11.3 SRAM memory throughput and latencies

The on-chip SRAM (OCRAM) itself does not have any latency once it is addressed to serve data. Similarly, random access does not add any additional penalty or latency. The only latency is on the NIC which will include read/write issue, read/write accept and the buffering:

1. Initial number clocks on the NIC. Minimum three for read and five for write.
2. Burst available. The NIC burst request required (invoked by cache or DMA), every clock 64-bit = 8B of data.

3.11.4 SDRAM throughput and latencies

Vybrid integrates a SDRAM controller that works on platform frequency (133-166 MHz). SDRAM is connected to the controller using 16-bit or 8-bit bus width. Maximum frequency is 400 MHz. The SDRAM controller can be synchronized to platform frequency only in ratio 2:1, which is not possible on Vybrid. This creates an additional penalty of five platform clocks to synchronize different clock domains prior to each transfer (worse latency, better throughput).

SDRAM controller supports burst mode — for each SDRAM clock, 4B are transferred (DDR per 16-bit bus width on 400 MHz). Data are stored in internal read and write buffers. The SDRAM memory controller is connected to the NIC via 64-bit bus width. Each transfer to the NIC in the case of burst can transfer 8B per platform clock (64-bit AXI bus on 133-166 MHz). However, 32B of data can be transferred in four platform clock cycles + initial clock cycles number.

Initial clock number depends on type of the SDRAM memory LP-DDR2/ DDR3 and the type of NIC buffering:

1. NIC master request (two-four platform clocks)
2. NIC slave accept (eight platform clocks - clock domain synchronization takes five platform clocks)
3. Buffering two-four platform clocks
4. Get the first byte from the SDRAM memory (seven SDRAM clocks, depends on the memory)
5. Access within page hit 6-1-1-1 SDRAM clocks

6. Random access: penalty if block miss (~10 SDRAM clocks). Access with page miss, up to (10+6)-1-1-1.

3.11.5 QuadSPI memory throughput and latencies

The QuadSPI controller is able to connect two identical memories in parallel mode. Maximum frequency of the controller is 104 MHz for SDR and 80 MHz for DDR. Despite the QuadSPI controller's ability to improve the speed, due to cooperation with wider range of memories, the guaranteed speed is only 25 MHz (30 MHz) DDR and 50 MHz SDR. However, we can transfer 8-bit/50 MHz using parallel mode. This is less than NIC node transfer ability. Due to this, QuadSPI integrates Rx and TX buffers. The QuadSPI controller supports 8, 16, 32 and 64-bit data bursts. Nevertheless, initial latency is as follows:

1. NIC master request (two-four platform clocks)
2. NIC slave accept (one platform clocks)
3. Buffering two-four platform clocks
4. Get the first bytes from the QuadSPI memory, three SPI clocks for DDR / five SPI clocks for SDR. The platform clock is 133 MHz-166 MHz. NIC: 64-bit/133 MHz = 8512 Mb/s; QSPI 8-bit/ 25 MHz = 200Mb/s => 42 clocks delay on NIC.

3.11.6 ARM cores in general

Applications can be optimized by using right opcodes. For example, ARM can use 32-bit opcodes which will include 16-bit code and 16-bit data (bus is 16 + 16). Latency kills the real-time, but not every instruction invokes load from memory since not every instruction can be processed in one clock cycle. For example, on average ARM7 needs ~1.9 cycles per instruction and ARM9 needs ~1.5 cycles per instruction.

4 Dual-core solutions

4.1 Configuration

Vybrid controller solutions offer different part/part numbers. Based on those numbers, you can get single or dual-core and Cortex-A5 or Cortex-M4 primary (booting) core – see Table 5.

Table 5. Vybrid cores configuration

num_cores[1:0]	Core Configuration
0	Cortex-A5 core only
1	Cortex-M4 core only
2	Dual core with Cortex-A5 as CP0 (boot core), Cortex-M4 as CP1
3	Dual core with Cortex-M4 as CP0 (boot core), Cortex-A5 as CP1

Number of cores can be checked in MSCM_CPxCOUNT register. Two core numbers exist. The physical core is defined by NIC node mumber in MSCM_CPxMASTER register. This number is constant: 0x02 for Cortex-A5 and 0x00 for Cortex_M4. The appropriate string can be read in MSCM_CPxType. For

programming, the logical core number is more important. The logical core number is stored in MSCM_CPxNUM register and is used for setting CPU to CPU interrupt, HW semaphores, etc. The logical number of the core depends on booting order. Booting the (primary) core always has logical number equal to zero, secondary core number equal to one.

4.1.1 Boot

- Boot ROM code starts at start address A5 0x0000_0000, M4 0x0001_6000.
- Boot ROM can set selected peripherals to enable boot from them, MMU can be enabled or disabled. Device configuration data (DCD) sets Vybrid to load the image from selected device and boot.

4.1.2 Memory configurations

There are many possible configurations of spreading the code and data. Examples of two of the configurations are presented for reference as follows:

- Vybrid memory configuration without external SDRAM usage where Cortex-M4 uses TCM with no latency, but smaller capacity.
  - Primary core Cortex-A5: 0x3f00_0000SRAM1 (256 kB)
  - Secondary core Cortex-M4: 0x1f80_0000TCML (32 kB)
  - Secondary core Cortex-M4: data0x3f80_0000TCMU (32 kB)
  - Shared memory: 0x3f04_0000SRAM2 (256 kB)
- Vybrid memory configuration with external SDRAM for extensive code and data needs.
  - Primary core Cortex-A5: 0x8000_0000SDRAM (MBs)
  - Secondary core Cortex-M4: 0x1f00_0000Alias to SRAM1 (128 kB)
  - Secondary core Cortex-M4: data 0x3f02_0000SRAM1 (128 kB)
  - Shared memory: 0x3f04_0000SRAM2 (256 kB)

4.1.3 Enabling a second core

The second core is not clocked after reset. The special purpose register in system reset controllers can hold an entry point for secondary core and an argument for entry function. Once set, the clock for a secondary core can be enabled. From that moment, both cores will run their own code.

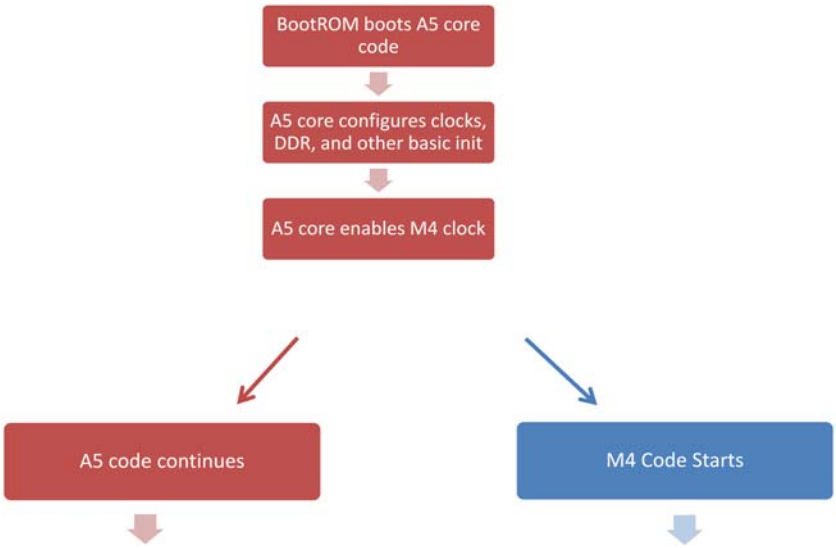

Figure 5. Dual-core start

An example of dual-core run with Cortex-A5 primary core is demonstrated in Figure 5. Second core enable steps:

- Set SRC_GPR2 -> Entry point for app to be executed on M4.
- Set SRC_GPR3 -> Any argument to pass to the function whose address is written in SRC_GPR2.
- Ensure that the address written in SRC_GPR2 is odd (THUMB Mode) as M4 supports only thumb mode.
- Enable auxiliary core clock: register CCM_CCOWR bit AUX_CORE_WKUP.

Code example:

```
printf("Hello World! from A5\n");
time_delay_ms(1000);
printf("Running second core \n");
printf("Setting start point \n");
SRC.GPR[2].R = 0x3f040410+1;
printf("Writing enable auxiliary clock\n");
CCM.CCOWR.R = 0x15a5a;
```

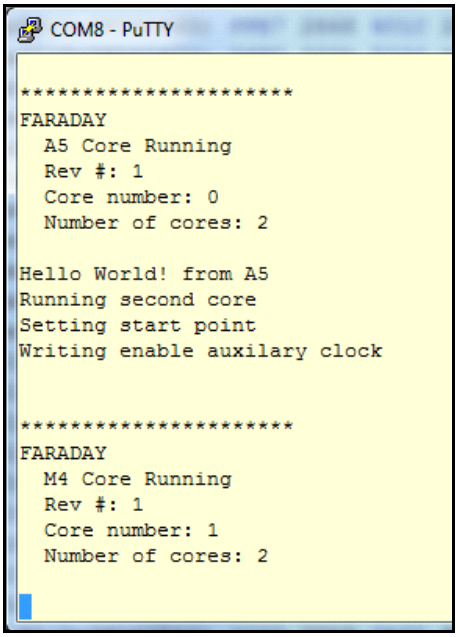

Figure 6. Dual-core start

Note that enabling clock for secondary core is done by write 0x…1…. to the CCM register. 0x….5a5a is the key.

4.1.4 Sharing peripherals

Two Cortex cores share a common memory map and 4 GB address space is effectively accessible from either core.

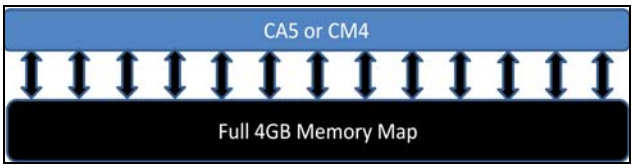

Figure 7. Dual-core shared space

Memory coherency is wholly managed by software with a few exceptions in the memory map:

- Cortex-M4: Tightly coupled memory (TCM) – (backdoor access)
- Cortex-M4: Aliased regions (DDR, QuadSPI, FlexBus, RAM)
- Cortex-A5: MMU, TrustZone

Take care about which peripherals are used by each core. All peripherals are mapped in a common address space. SCI is used in the previous example. Both cores are writing to the same data register. If not synchronized, write to data buffer can stop current transmit. Vybrid controller solutions support a couple of HW tools described in following paragraph. Moreover, we can protect memory regions via the central security unit (CSU), TrustZone controller (AHB-TZASC), or lock NIC connection. More information can be found in *Vybrid Reference Manual* [2] and *Vybrid Security Reference Manual* [7].

4.2 Dual-core communication and synchronization options

Vybrid controller solutions support:

- HW semaphores can be locked by one core only at the time. Only the same core can unlock it.
- Interrupts. Routing options for both cores — none, one, or both cores can be notified for the event.
- CPU to CPU interrupts — a direct event for both or the other core.
- Shared memory — any memory (preferably SRAM) protected by HW semaphores.
- MCC protocol — specially designed for Vybrid controller solutions for Linux® and MQX™ or MQX and MQX cooperation.

5 Latency measurement

The best way to get latency on each peripheral is to measure it. For measuring latency, we need a hardware timer. A standard FlexTimer is connected on the peripheral bus, which involves transfer through the NIC, adding latency. To measure it right, we have to use the internal timer that is part of the ARM core. Cortex-A5 and Cortex-M4 use a different access to the built-in timer. Tests are performed using The Freescale Tower System module (TWR-VF65GS10).

5.1 Cortex-M4 memory latency measurement

For latency measurement on Cortex-M4, a simple increment instruction on global variable was used:

```
uint32 memory_test = 0;

memory_test++;
```

To measure the execution time, SysTick was used. The SysTick is controlled by CSR, RVR and CVR core registers. SysTick runs on platform clock. The method is using linear code — see Figure 7. No wait state is considered due to pipelining. The execution is more complex due to pipelining — each instruction has three stages (fetch decode, execution). STR R0,[R4,#08] instruction starts the SysTick timer. The instructions LDR, ADDS, and STR increment the variable located in selected memory. They are executed according to ARM specification in 2, 1 and 2 Cortex-M4 core cycles — a total of five core cycles. LDR R0,[R4,#0x4] copies the SysTick counter value into the core register R0, which takes one additional clock cycle due to LDR instruction execution time. Together, it takes six clock cycles to increment the variable and store the value of the SysTick counter in the case that there is no additional latency depending on the variable memory location. For more information about instruction execution, see *Cortex-M4 Processor Technical Reference Manual* [8] chapter “Load/store Timings.”

```
    SYSTICK_Reset ();
??main_1:
   0x3f040e70: 0x2000          MOVS      R0, #0
   0x3f040e72: 0x60a0          STR       R0, [R4, #0x8]
   memory_test++;
   0x3f040e74: 0x6828          LDR       R0, [R5]
   0x3f040e76: 0x1c40          ADDS      R0, R0, #1
   0x3f040e78: 0x6028          STR       R0, [R5]
    u32NumOfCycles = SYSTICK_GetTicks();
   0x3f040e7a: 0x6860          LDR       R0, [R4, #0x4]
   0x3f040e7c: 0x68a1          LDR       R1, [R4, #0x8]
   0x3f040e7e: 0x1a40          SUBS      R0, R0, R1
   0x3f040e80: 0x6068          STR       R0, [R5, #0x4]
```

Figure 8. Cortex-M4 latency measurement code

5.2 Cortex-M4 use cases results

Each use case represents a different code/data memory location configuration. The IAR Workbench 6.50.3.4757, components 6.5.5.2622, was used. Each use case represents one line in Table 6. The Cortex-M4 L1 cache default setting in the local memory controller was used.

Table 6. Cortex-M4 latency measurement results

Use case	CODE	DATA	LMEM CODE CACHE	LMEM SYSTEM CACHE	Execution		Wait states	
					first	subsequent	first	subsequent
1	TCML (CODE BUS)	TCMU (SYSTEM BUS)	enabled	enabled	6	6	0	0
2	TCML (CODE BUS)	TCMU (SYSTEM BUS)	enabled	disabled	6	6	0	0
3	S3 (CODE BUS)	S4 (SYSTEM BUS)	enabled	enabled	7	6	1	0
4	S3 (SYSTEM BUS)	S4 (SYSTEM BUS)	enabled	enabled	8	7	2	1
5	S3 (CODE BUS)	S4 (SYSTEM BUS)	enabled	disabled	18	18	12	12
6	TCML (CODE BUS)	S3 (SYSTEM BUS)	x	enabled	6	6	0	0
7	TCML (CODE BUS)	S3 (SYSTEM BUS)	x	disabled	18	18	12	12
8	TCML (CODE BUS)	S4 (SYSTEM BUS)	x	enabled	6	6	0	0
9	TCML (CODE BUS)	S4 (SYSTEM BUS)	x	disabled	18	18	12	12

Note the following:

- TCML (CODE BUS) is tightly coupled memory located on 0x1f800000;
- TCML (SYSTEM BUS) is tightly coupled memory located on 0x3f800000;
- S3 (CODE BUS) is on-chip SRAM (The NIC S3 node) code alias on 0x1f000000; (write-through cache)
- S3 (SYSTEM BUS) is on-chip SRAM (The NIC S3 node) located on 0x3f000000; (write-back cache)
- S4 (SYSTEM BUS) is on-chip SRAM (The NIC S4 node) located on 0x3f040000; (write-back cache)

Cortex-M4 results:

- **Use cases 1 and 2:** When TCM is used, L1 cache usage does not have any effect on the execution time. Execution time is always six clock cycles — TCM does not have any latency.
- **Use case 3:** The code and the data are located in on-chip SRAM memory. Instructions/data have to go though the NIC. Caches are enabled. The first execution requires an additional clock cycle due to NIC latency. Instruction/data are not cached yet. In subsequent execution instruction/data are already cached — no additional latency.
- **Use case 4:** In this case, a code alias is not used. The core fetches instructions via system bus with one wait state.
- **Use case 5:** The system cache is not used — accessing non-cached data through the NIC takes 12 additional clock cycles.
- **Use case 6-9**: The instructions are located in TCML (code bus) — accessing non-cached data through the NIC requires 12 additional clock cycles. The on-chip SRAM 256 KB memory banks on S3 and S4 NIC nodes are equivalent.

5.3 Cortex-A5 memory latency measurement

For latency measurement on Cortex-A5 simple increment instruction on global variable, two simple tests were used:

```
uint32 memory_test = 0;

memory_test++;
```

And the simplest one:

```
asm("nop");
```

To measure the execution time, the Cycle Count Register (PMCCNTR) was used. The PMCCNTR counts processor clock cycles and is controlled by the User Enabled Register.

```
PMCCNTR_Reset:
   0x3f0014d8: 0x2005          MOVS      R0, #5
  asm("MCR p15, 0, r0, c9, c12, 0");
   0x3f0014da: 0xee09 0x0f1c  MCR       p15, #0, R0, C9, C12
}
   0x3f0014de: 0x4770          BX        LR
  asm("MOVS R0, #0x4");
PMCCNTR_Disable:
   0x3f0014e0: 0x2004          MOVS      R0, #4
  asm("MCR p15, 0, r0, c9, c12, 0");
   0x3f0014e2: 0xee09 0x0f1c  MCR       p15, #0, R0, C9, C12
}
   0x3f0014e6: 0x4770          BX        LR
  register uint32 u32R0 = 0;
PMCCNTR_GetTicks:
   0x3f0014e8: 0x2000          MOVS      R0, #0
  asm("MRC p15, 0, r0, c9, c13, 0");
   0x3f0014ea: 0xee19 0x0f1d  MRC       p15, #0, R0, C9, C13
  return u32R0;
   0x3f0014ee: 0x4770          BX        LR
int main()
```

Figure 9. Cortex-A5 PMCCNTR code

The PMCCNTR control code is shown in Figure 9. MCR p15, #0, R0, c9, c12 commands represent write to the counter register. If R0 is #5, then the counter is reset. MRC p15, #0, R0, c9, c12 commands represent read value of the counter register. For more information, see the *Cortex-A5 MPCore Technical Reference Manual* [9], chapter “Cycle Count Register.”

```
   0x3f001522: 0xf7ff 0xffd1  BL        PMCCNTR_Enable    ...
  _dcache_invalidate();
   0x3f001526: 0xf000 0xf9ab  BL        _dcache_invalidate...
  _icache_invalidate();
   0x3f00152a: 0xf000 0xf9ad  BL        _icache_invalidate...
  PMCCNTR_Reset ();
   0x3f00152e: 0xf7ff 0xffd3  BL        PMCCNTR_Reset     ...
   memory_test++;
   0x3f001532: 0x4c2f          LDR.N     R4, ??DataTable8_3...
   0x3f001534: 0x6820          LDR       R0, [R4]
   0x3f001536: 0x1c40          ADDS      R0, R0, #1
   0x3f001538: 0x6020          STR       R0, [R4]
  u32NumOfCycles = PMCCNTR_GetTicks();
   0x3f00153a: 0xf7ff 0xffd5  BL        PMCCNTR_GetTicks  ...
   0x3f00153e: 0xf20f 0x05dc  ADR.W     R5, ?<Constant "A5...
   0x3f001542: 0x6060          STR       R0, [R4, #0x4]
  printf("A5: Data read and write cycles: %d\n",u32NumOfCy...
   0x3f001544: 0x6861          LDR       R1, [R4, #0x4]
   0x3f001546: 0x4628          MOV       R0, R5
   0x3f001548: 0xf7ff 0xfab8  BL        printf            ...
  for (i=0;i<10;i++) {
```

Figure 10. Cortex-A5 latency measurement code

The measuring method is similar to the previous method used with Cortex-M4, except using the functions for PMCCNTR counter control instead of macro usage. The linear code is also used — see Figure 10. Prior to each measurement, the module is reset (POR) right before measurement caches are invalidated and PMCCNTR is reset. The instructions LDR.N, LDR, ADDS and STR increment the variable located in selected memory. They are executed according to ARM specification in LDR.N 2 clocks green line; LDR 1, ADDS 1, STR 2 clocks; Cortex-A5 core cycles — a total of six core cycles.

Handling PMCCNTR counter requires additional clock cycles: BL (to PMCCNTR_GetTicks) takes one clock, MOVS R0,#0 takes one clock, MCR p15 takes one clock. In sum, it is eight clock cycles due to

Cortex-A5 optimization in the case that there is no additional latency depending on the variable memory location.

5.4 Cortex-A5 use cases results

Each defined use case represents different cache setting and code/data memory location configuration, as indicated in entries in the following tables. All measurements include three PMCCNTR control clock cycles. Data in the table present Cortex-A5 clock cycles, which are three times faster than the platform clock (Cortex-M4 clock).

Table 7. Cortex-A5 latency measurement results for memory_test ++ test

	L2	L1 I	L1 D	TLB	S3 / SRAM		S5 / GRAM		S8 / SDRAM		S2/S7 /QSPI 18MHz	
					First	Subsequent	First	Subsequent	First	Subsequent	First	Subsequent
1	0	0	0	off	96	75	132	75	437	371	4363	2770
2	0	0	0	on	85	43	93	43	345	269	4408	2770
3	0	0	1	on	129	8	103	9	342	263	5407	2770
4	0	1	1	on	129	8	103	9	264	8	4888	9
5	1	1	1	on	147	8	115	9	285	8	4888	9

Cortex-A5 memory_test++ test results comments:

Use case 1: In this case, no cache or MMU is used. Due to three times faster clock and attempt to access memory without cache, both run; first run and subsequent run add high latency. Each instruction data have to go through L2 cache controller, NIC, buffering, clock domain change, and then to selected memory. Communication with SDRAM memory takes additional latency for synchronization and additional latency to jump for given block and page. This is a similar situation to QuadSPI, which is even slower due to lower frequency (tested on 18 MHz) and bus width (8-bit). Some commands have to be transferred prior to read first byte of data using 8-bit bus width in parallel mode. Cortex-A5 capability and NIC buffer improve performance on subsequent run.

Use case 2: Once MMU (TLB) is enabled, the special features of Cortex-A5 (like predictions and speculation) are enabled, which decrease latency. Explanation of those features is out of the scope of this AN due to its complexity.

Use case 3: When L1 data cache is enabled it significantly improves performance on subsequent run. The first run is slower (cache load).

Use case 4. Adding L1 code cache significantly reduces latency on SDRAM and QuadSPI on subsequent run due to reduction of direct code/data transfers from/to the memory.

Use case 5. Enabling L2 cache does not have impact because the program is short. For higher code/data density, L2 cache is very important.

Table 8. Cortex-A5 latency measurement results for NOP test

	L2	L1 I	L1 D	TLB	S3 / SRAM		S5 / GRAM		S8 / SDRAM		S2/S7 /QSPI 18MHz	
					First	Subsequent	First	Subsequent	First	Subsequent	First	Subsequent
1	0	0	0	off	54	19	57	19	371	365	4726	2407
2	0	0	0	on	44	4	53	4	287	263	3226	2407
3	0	0	1	on	64	10 (7)	54	7	287	263	3499	2407
4	0	1	1	on	64	10 (7)	54	7	181	7	4045	7
5	1	1	1	on	70	10 (7)	60	7	190	7	4045	7

Cortex-A5 NOP test results comments:

Table 8 represents equivalent results to previous memory_test++ test. In the test no data access is required. The first execution is shorter and the cached subsequent executions are two clocks shorter.

6 Conclusion

This application note presents the details of the Vybrid asymmetrical multiprocessing architecture. Users will gain a better understanding of architecture features and limitations to achieve real-time performance by selecting the right core, memory, and cache setting for the selected application. The presented data indicate that, for example, the Cortex-M4 core can achieve better results together with TCM than Cortex-A5 running on three times higher frequency, especially when running code that is not in the cache or in the case of liner code because latency kills the real time. This is extremely important in real-time applications like safety systems or motor control applications. Presenting measured data of latency in selected use cases can assist the customer to select the most appropriate option for their application.

7 References

1. Vybrid ARM Controller Solution: freescale.com/vybrid
2. Vybrid Reference manual (VYBRIDRM)
   http://cache.freescale.com/files/32bit/doc/ref_manual/VYBRIDRM.pdf?fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation
3. Vybrid Datasheet (VYBRIDFSERIESEC)
   http://cache.freescale.com/files/microcontrollers/doc/data_sheet/VYBRIDFSERIESEC.pdf?pspll=1
4. [4] Vybrid tower module TWR-VF65GS10
   http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=TWR-VF65GS10&fsrch=1
5. [5] NIC-301 Technical Reference Manual, ARM infocenter.arm.com
   http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0397i/index.html
6. [6] Cortex-A Series Programmer's Guide
   https://silver.arm.com/download/download.tm?pv=1550621

7. [7] Vybrid Security reference manual (VYBRIDSRM)
http://www.freescale.com/webapp/sps/download/mod_download.jsp?colCode=VYBRIDSRM&location=null&appType=moderated&fpsp=1&WT_TYPE=Reference%20Manuals&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation&Parent_nodeId=133106782916669711663 0&Parent_pageType=product
8. [8] ARM Cortex-M4 Processor Technical Reference Manual
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0439d/CHDIJAFG.html
9. [9] Cortex-A5 MPCore Technical Reference Manual
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0434b/index.html

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4947
Rev. 0
07/2014