

AURIX 2G Multi-Core Features and programming

Hansen Chen
IFCN ATV SMD GC SAE MC
2018/5/2



Agenda

- 1 AURIX 2G Family and Multi-core Architecture
- 2 Memory Map, Code and Data allocation
- 3 Associate Core, Data and Code in Tasking
- 4 Associate Core, Data and Code in GNU
- 5 Some common issues in Multi-core system



Agenda

- 1 **AURIX 2G Family and Multi-core Architecture**
- 2 Memory Map, Code and Data allocation
- 3 Associate Core, Data and Code in Tasking
- 4 Associate Core, Data and Code in GNU
- 5 Some common issues in Multi-core system

AURIX™ TC3xx

From low cost to high performance applications



9x Series up to 16 MB						TC397Xx TC397Qx 300MHz	TC399Xx 300MHz	MCU Scalability <ul style="list-style-type: none"> › Performance & Flash › Pin-compatibility › Binary compatible cores
8x Series up to 10MB						TC387Q 300MHz	TC389Q 300MHz	
7xA Series up to 6MB						TC377TX 300 MHz		
7x Series up to 6MB				TC375T 300 MHz		TC377T 300 MHz		Power Consumption <ul style="list-style-type: none"> › On-chip SC DC/DC high-efficiency power supply
6x Series up to 4MB		TC364D 300 MHz	TC364D 300 MHz	TC365D 300 MHz	TC366D 300 MHz	TC367D 300 MHz		
5xA Series up to 4MB					TC356TA 300 MHz	TC357TA 300 MHz		
3xA Series up to 2 MB			TC334DA 200 MHz		TC336DA 200 MHz	TC337DA 200 MHz		Safety/Security Concept <ul style="list-style-type: none"> › ISO26262 compliance › Hardware security support  
3x Series up to 2 MB	TC333L 200 MHz	TC334L 200 MHz	TC334L 200 MHz		TC336L 200 MHz	TC337L 200 MHz		
2x Series up to 1 MB	TC323L 160 MHz	TC324L 160 MHz				TC327L 160 MHz		
Flash								Connectivity <ul style="list-style-type: none"> › Ethernet: Up to 2x 1 Gigabit › CAN FD: up to 12 channels › LIN: up to 24 channels › eMMC IF
Package	TQFP 100	T/LQFP 144	BGA144*	LQFP 176	LBGA 196	LFBGA 292	LFBGA 516	

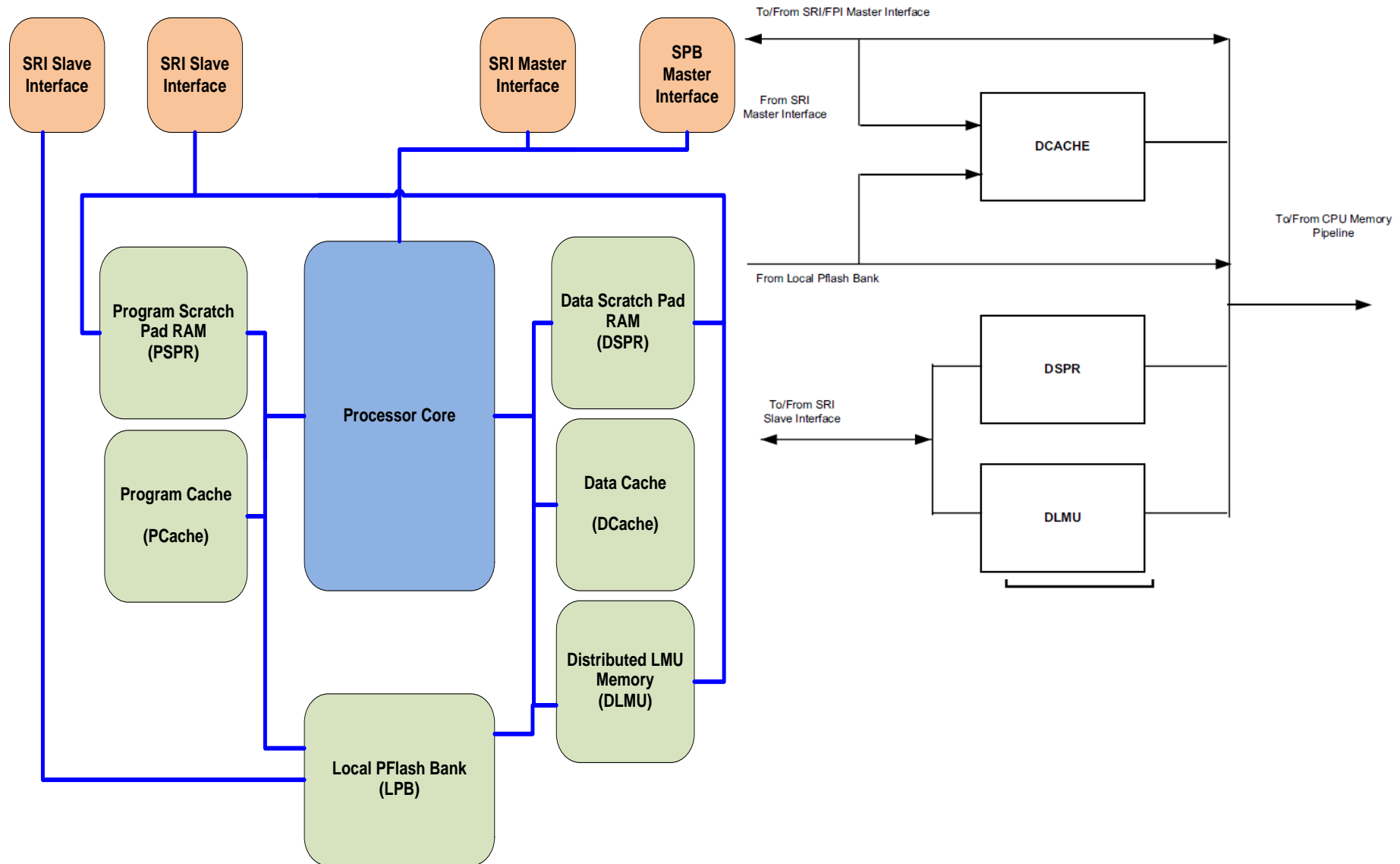
L - Single Lockstep Core
 D - Dual Core
 T - Triple Core
 Q - Quadruple Core
 X - Sextuple Core

AURIX TC3xx Feature Table

Feature Set		9x Series eXtension (16MB)	8x Series (10MB)	7x Series eXtended (6MB)	7x Series (6MB)	6x Series (4MB)	5x Series (4MB)	3x Series +eXtension (2MB)	3x Series (2MB)	2x Series (1MB)
TriCore 1.6	# Cores / Checker	6/4	4/2	3/3	3/2	2/2	3/2	2/1	1/1	1/1
	Frequency	300MHz	300MHz	300MHz	300MHz	300MHz	300MHz	200MHz	200MHz	160MHz
Accelerator	Signal processing Unit (SPU)	2xSPU					2xSPU	1xSPU		
Flash	Program Flash	16MB	10MB	MB	6MB	4MB	4MB	2MB	2MB	1MB
	Data Flash (physical/logical)	1024kB	512kB	256kB	256kB	128kB	128kB	128kB	128kB	96kB
SRAM	Total (DMI , PMI, LMU, AMU)	6912KB	1568kB	4208kB	1136kB	672kB	2837kB	1328kB	248kB	152kB
DMA	Channels	128	128	128	128	64	64	16	16	16
ADC	Modules Primary / Sec / FC / DS	8/4/8/14	8/4/4/10	4/4/2/6	4/4/2/6	4/2/2/4	2/0/0/0	4/2/0/0	2/2/0/0	2/2/0/0
	Channels Primary / Sec / FC / DS	64/64/8/14	64/64/4/10	32/64/2/6	32/64/2/6	32/32/2/4	16/0/0/0	>16/32/0/0	16/32/0/0	16/32/0/0
Timer	GTM TIM / (A)TOM / MCS	64 / 192 / 10	56 / 152 / 7	40 / 88 / 5	40 / 88 / 5	24 / 64 / 3	-	16 / 32 / 0	16 / 32 / 0	16 / 32 / 0
	CCU / GPT modules / bit streaming	2/1/1	2/1/0	2/1/0	2/1/-	2/1/0	2/1/1	2/1/1	2/1/0	2/1/0
Interfaces	FlexRay (#/ch.)	2 /4	2/4	1/2	1/2	1/2	1/2	1/2	1/2	0/0
	CAN-FD / TT	12/1	12/1	12*/1	8/1	8/1	6/0	8/0	8/0	6/0
	QSPI / ASCLIN / I2C	6 /12/2	5 /24/2	5/12/1	5/12/1	4/12/1	4/4/0	4/12/0	4/12/0	4/6/0
	SENT / PSIS / PSIS5	25/4/1	25/4/1	15/2/1	15/2/1	10/2/1	0/0/0	6/0/0	6/0/0	6/0/0
	HSSL / MSC / EBU	2/4/1	1/3/0	1/2/0	1/2/0	1/1/0	0/0/0	0/0/0	0/0/0	0/0/0
	Ethernet 100Mbps/1Gbps	1/1	1/1	2*/2*	1/1	1/1	1/1	1/1	-/-	-/-
	eMMC/SDIO	1/1		1/1				1/1		
	Radar /ext. ADC IF (RIF)	12x400Mbps LVDS	-	-	-	-	12x400Mbps LVDS	6x100Mbps LVDS	-	-
	Camera IF (CIF)	-	-	1	-	-	-	-	-	-
Security	HSM	HSM+ECC256	HSM+ECC256	HSM+ECC256	HSM+ECC256	HSM+ECC256	HSM+ECC256	HSM+ECC256	HSM+ECC256	HSM+ECC256
Safety	SIL Level	ASIL D	ASIL D	ASIL D	ASIL D	ASIL D	ASIL D	ASIL D	ASIL D	ASIL D
Power	EVR	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)	Yes (3.3V/5V)
	Standby Control Unit	yes	yes	yes	yes	yes	yes	yes	yes	yes

**in discussion*

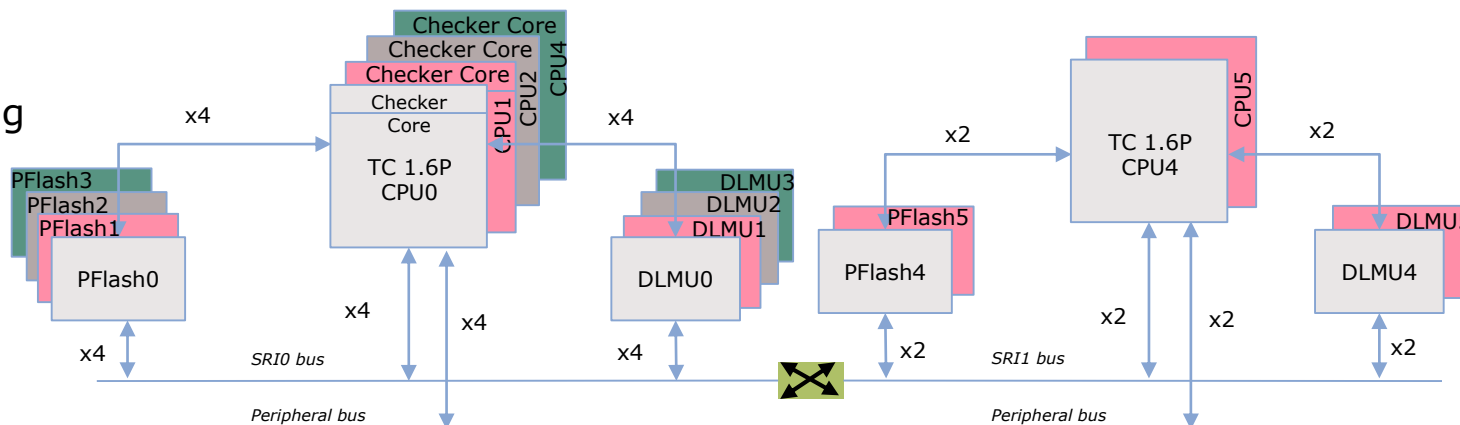
Processor Core, Local Memory and connectivity



AURIX 2G Multi-Core Architecture

Hexa Core/Quad
Lockstep

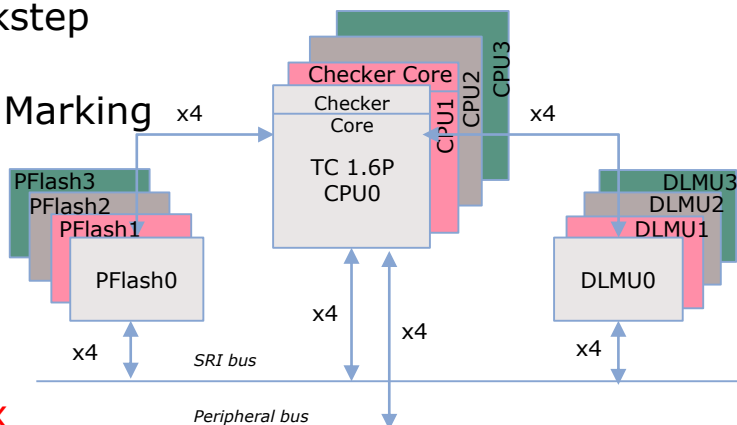
-
"X" Marking



TC39x

Quad Core/ Dual
Lockstep

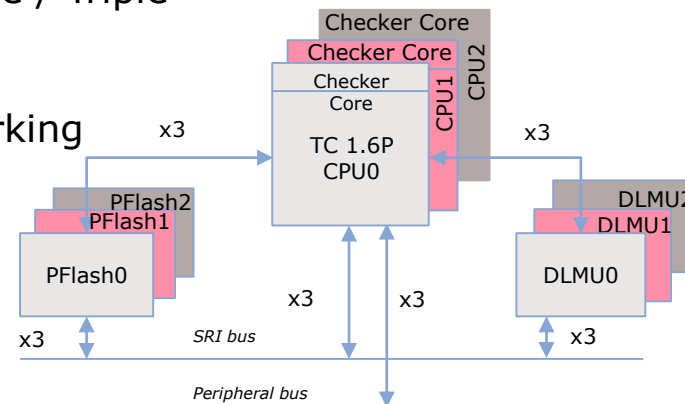
-
"Q" Marking



TC38x

Triple core / Triple
Lockstep

-
"TX" Marking

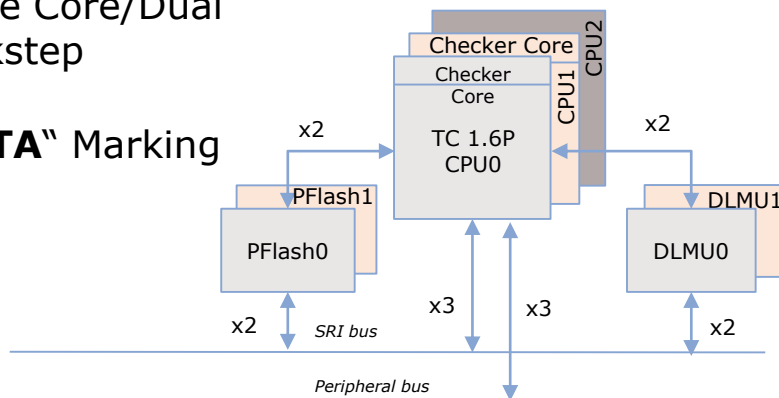


TC37x Extension

AURIX 2G Multi-Core Architecture(cont.)

Triple Core/Dual Lockstep

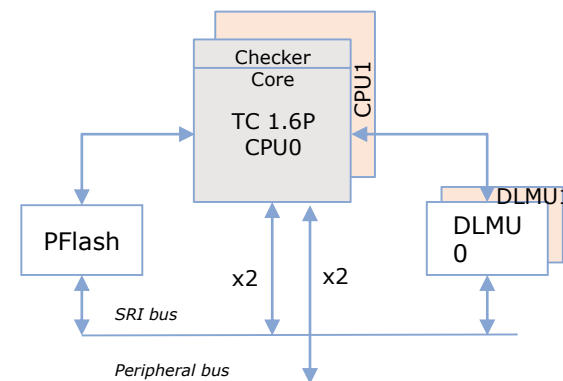
-
"T/TA" Marking



TC37x and TC35x

Dual Core/ One Lockstep

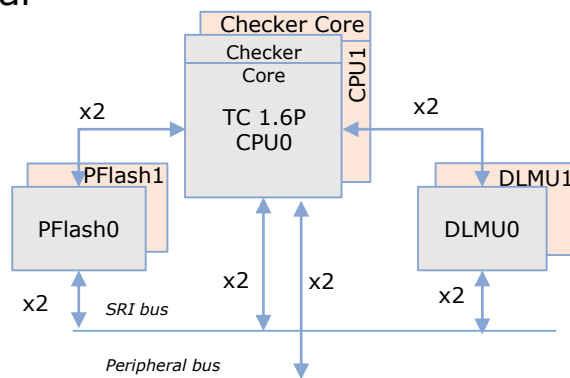
-
"DA" Marking



TC33x Extension

Dual Core/ Dual Lockstep

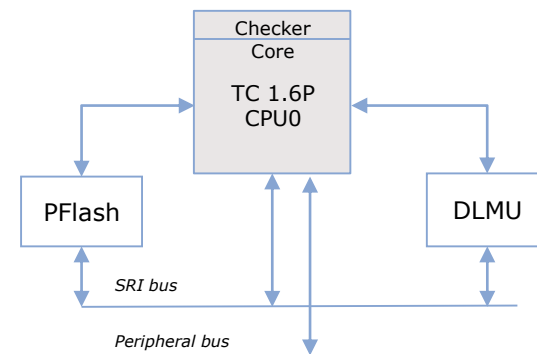
-
"D" Marking



TC36x

One core/One Lockstep

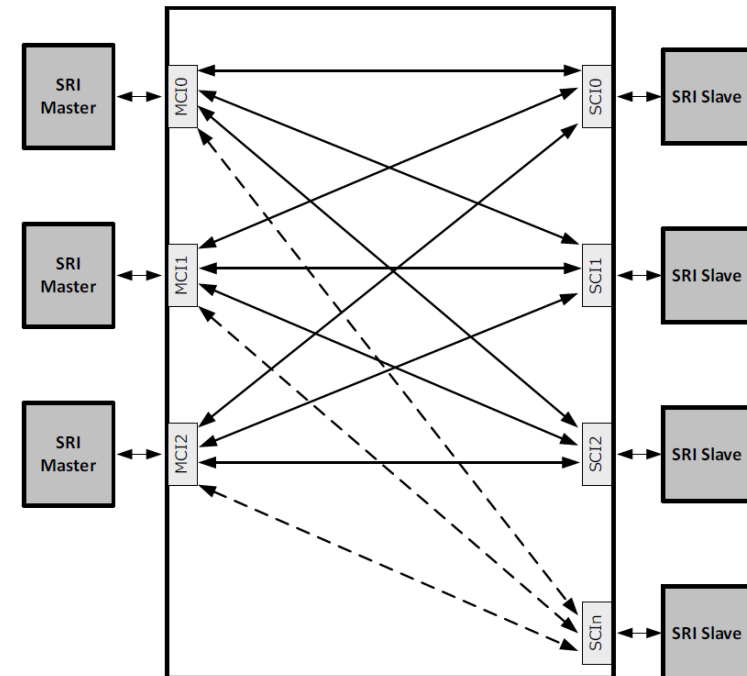
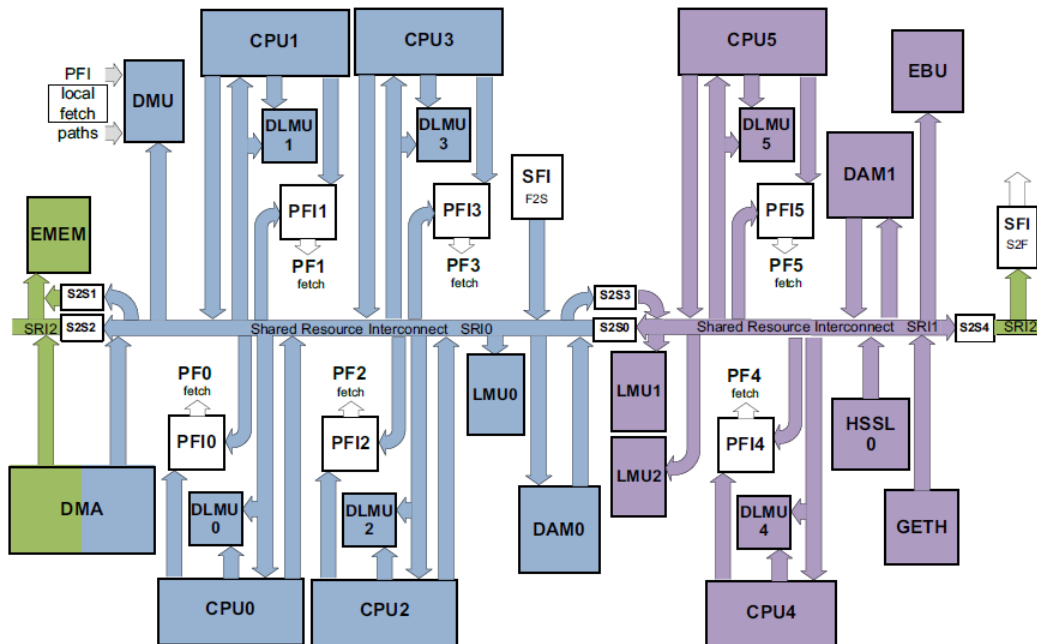
-
"L" Marking



TC33xL, TC32xL

SRI Domain and Crossbar for Multi-core

- › One Master has direct access to several slave ports
- › One slave serves one master at the same time
- › Access Latency still can happen, if several masters using the same slave
- › There is no access latency when different slaves are addressed
- › Code and Data location has great influence on bus performance



New features in SRI

- › Major differences of the AURIX 2G vs AURIX 1G.
 - SRI Fabric can now consist of one or multiple independent crossbars
 - SRI Crossbar arbitration scheme simplified to two layer round-robin
 - a high priority level and a low priority level (SCIx Arbiter Priority Register: **PRIORITYx**)
 - Question : what is the difference to AURIX 1G?

Mapping of AURIX_{TM} SRI Masters to Domain 0 MCI

Derivative	MCI0	MCI1	MCI2	MCI3	MCI4	MCI5	MCI6	MCI7	MCI8	MCI9	MCI10	MCI11	MCI12
TC39xA	DMA MIFO	SFI F2S	CPU0	CPU1	CPU2	CPU3	S2S0 D1D0	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present
TC39xB	DMA MIFO	SFI F2S	CPU0	CPU1	CPU2	CPU3	S2S0 D1D0	Not Present	HSSLO	Not Present	Not Present	Not Present	Not Present
TC38x	DMA MIFO	SFI F2S	CPU0	CPU1	CPU2	CPU3	Not Present	Not Present	HSSLO	GETH	Not Present	Not Present	Not Present
TC37xED	DMA MIFO	SFI F2S	CPU0	CPU1	CPU2	Not Present	Not Present	Not Present	HSSLO	GETH	Not Present	Not Present	GETH1
TC37x	DMA MIFO	SFI F2S	CPU0	CPU1	CPU2	Not Present	Not Present	Not Present	HSSLO	GETH	Not Present	Not Present	Not Present
TC36x	DMA MIFO	SFI F2S	CPU0	CPU1	Not Present	Not Present	Not Present	Not Present	HSSLO	GETH	Not Present	Not Present	Not Present
TC35x	DMA MIFO	SFI F2S	CPU0	CPU1	CPU2	Not Present	Not Present	Not Present	Not Present	GETH	DMA MIF1	DMA MIF2	Not Present
TC33xED	DMA MIFO	SFI F2S	CPU0	CPU1	Not Present	Not Present	Not Present	Not Present	Not Present	GETH	Not Present	Not Present	Not Present
TC33x	DMA MIFO	SFI F2S	CPU0	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present

Mapping of AURIX_{TM} SRI Slaves to Domain 0 SCI

Derivative	SCI0	SCI1	SCI2	SCI3	SCI4	SCI5	SCI6	SCI7	SCI8	SCI9	SCI10	SCI11	SCI12	SCI13	SCI14	SCI15
TC39xA	S2S3 D0D1	DMU	DAM0	CPU0P	CPU0S	CPU1P	CPU1S	CPU2P	CPU2S	CPU3P	CPU3S	LMU0	S2S2 D0D2	S2S1 D0D2	Not Present	Default Slave
TC39xB	S2S3 D0D1	DMU	DAM0	CPU0P	CPU0S	CPU1P	CPU1S	CPU2P	CPU2S	CPU3P	CPU3S	LMU0	S2S2 D0D2	S2S1 D0D2	Not Present	Default Slave
TC38x	Not Present	DMU	DAM0	CPU0P	CPU0S	CPU1P	CPU1S	CPU2P	CPU2S	CPU3P	CPU3S	LMU0	Not Present	Not Present	Mini MCDS	Default Slave
TC37xED	Not Present	DMU	DAM0	CPU0P	CPU0S	CPU1P	CPU1S	Not Present	CPU2S	Not Present	Not Present	Not Present	S2S0 D0D2	S2S1 D0D2	Not Present	Default Slave
TC37x	Not Present	DMU	DAM0	CPU0P	CPU0S	CPU1P	CPU1S	Not Present	CPU2S	Not Present	Not Present	Not Present	Not Present	Not Present	Mini MCDS	Default Slave
TC36x	Not Present	DMU	DAM0	CPU0P	CPU0S	CPU1P	CPU1S	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Default Slave
TC35x	SFI S2F BBB	DMU	Not Present	CPU0P	CPU0S	CPU1P	CPU1S	Not Present	CPU2S	EMEM 0	EMEM 1	LMU0	LMU1	Not Present	Not Present	Default Slave
TC33xED	SFI S2F BBB	DMU	Not Present	CPU0P	CPU0S	Not Present	CPU1S	Not Present	Not Present	EMEM 0	Not Present	Not Present	Not Present	Not Present	Not Present	Default Slave
TC33x	Not Present	DMU	Not Present	CPU0P	CPU0S	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Not Present	Default Slave

AURIX™ TC3xx Memory Concept

Local And Global Memory



- › Enhanced protection in AURIX™ - TC3xx
All memory data and addresses and all accesses to all memories are protected
- › Enhanced local and global memory concept
 - Local memories are providing enhanced performance to the specified CPU: PSPR, DSPR, dLMU, PFLASH, (Data Cache, Program Cache)
 - All CPUs can access all local memories of other CPUs with "global" performance
 - Global memories have no preferences: global LMU, DAM, EMEM, D-Flash
 All CPUs can access all global memories with "global" performance
- › memory access performance

local access wait states

access type	data read	data write	code fetch
local PSPR	6	7	0
local DSPR	0	0	
local dLMU	1	2	7
local PFLASH	3+Flash		2+Flash

global access wait states

access type	data read	data write	code fetch
other PSPR	6	7	6
other DSPR	6	6	6
other local dLMU	7	8	7
other PFLASH	9+Flash		8+Flash
DFLASH	8+Flash		
global LMU, DAM	7	8	7
EMEM	18	10	18

Agenda

- 1 AURIX 2G Family and Multi-core Architecture
- 2 **Memory Map, Code and Data allocation**
- 3 Associate Core, Data and Code in Tasking
- 4 Associate Core, Data and Code in GNU
- 5 Some common issues in Multi-core system

AURIX 2G Address Map

- › TriCore 4GB Addressable Memory Map is organized into Segment of 256 MBytes.
- › A Segment is identified by the A[31:28] bits of the system address
- › Each Segment allows access to specific area location. Also used to define cacheable and non-cacheable areas
- › System Address Map is defined in the chapter 'Memory Map'
- › Additional / private views are described in the module chapter (e.g. CPU segment C/D)
- › **Segments 0&2** : Reserved
- › **Segment 1 and 3-7**: Access to the CPUs Program and Data Scratch Pad SRAM
- › **Segment 8**: Cached Access to Pflash & BROM
- › **Segment 9**: Cached Access to LMU & EMEM
- › **Segment 10**: Non-cached access to PFlash, DFlash, BROM
- › **Segment 11**: Non-cached access to LMU & EMEM
- › **Segment 12-13**: reserved
- › **Segment 14**: Reserved
- › **Segment 15**: The lower 128 Mbyte is SPB address space and the upper 128 Mbyte is SRI address space

AURIX 2G Multi-Core Address Map

Local Memories - 1



Each CPU has access to its own memory by:

› **Private address**

- Code Scratch memory at local address 0xC000, 0000
- Data Scratch memory at local address 0xD000, 0000

› **Public address** via its global segment (1,3-7)

- Data Scratch Memory at offset 0
- Code Scratch Memory at offset 0x0010 0000 (1MB)

This provides:

- › The best usage of TriCore addressing modes for optimum code-density (single instruction access).
- › The unique address for CPU SRAMs in the system

Segment Number

0xD CPU0..7 Data
Scratch memory

0xC CPU0..7 Code
Scratch memory

0x7	CPU0 Code+Data Scratch
0x6	CPU1 Code+Data Scratch
0x5	CPU2 Code+Data Scratch
0x4	CPU3 Code+Data Scratch
0x3	CPU4 Code+Data Scratch
0x2	reserved
0x1	CPU5 Code+Data Scratch
0x0	reserved

AURIX 2G Multi-Core Address Map

Local Memories - 2



- › Physical Memory Attribute Registers (PMA) control the cached/non-cached behaviour of Code Fetch and Load/Store accesses (individually) to non-local CPU memories
- › Data access to local DSPR, local DLMU are non-cached
- › Instruction fetch from local PSPR are non-cached
- › Access to memories of other CPUs can be cached or non-cached (control per CPU via Memory Attribute)
- › All SRAMs can be used as Overlay SRAM
 - any DSPR SRAM, EMEM, LMU SRAM, external memory

Segment Number

0xD	CPU0..7 Data Scratch memory
-----	--------------------------------

0xC	CPU0..7 Code Scratch memory
-----	--------------------------------

0x7	CPU0 Code+Data Scratch
0x6	CPU1 Code+Data Scratch
0x5	CPU2 Code+Data Scratch
0x4	CPU3 Code+Data Scratch
0x3	CPU4 Code+Data Scratch
0x2	reserved
0x1	CPU5 Code+Data Scratch
0x0	reserved

AURIX 2G Multi-Core Address Map

Local Memories - 3

> CPUx Data Access to

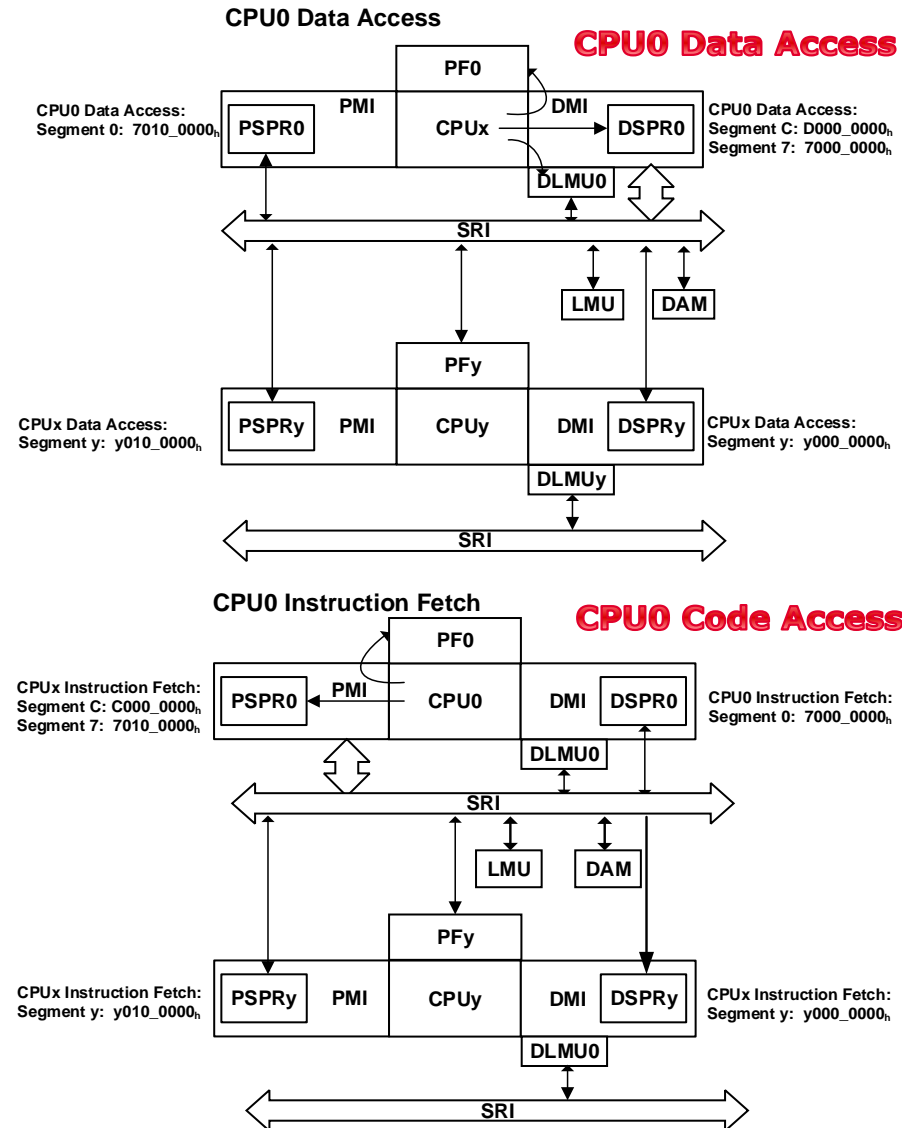
- Local DMI: 0 penalty
- Local Pflash Bank :
 - Access via local interface : 3+flash wait state
 - Access via XBar_SRI
- Local DLMU: 1 penalty
- Local PMI: Own PSPR via XBAR_SRI

> CPUx Code Fetch from

- Local PMI: 0 penalty
- Local Pflash Bank:
 - Access via local interface : 2 +flash wait state
 - Access via XBar_SRI
- Local DMI : Own DSPR via XBAR_SRI
- Local DLMU: Own DLMU via XBAR_SRI

> Always via **XBar_SRI**:

- Non-Own DSPR/PSPR/LMU/DAM/PFLASH and DFLASH
- EMEM access
- External memories



AURIX 2G Multi-Core Address Map

PFLASH and DFLASH

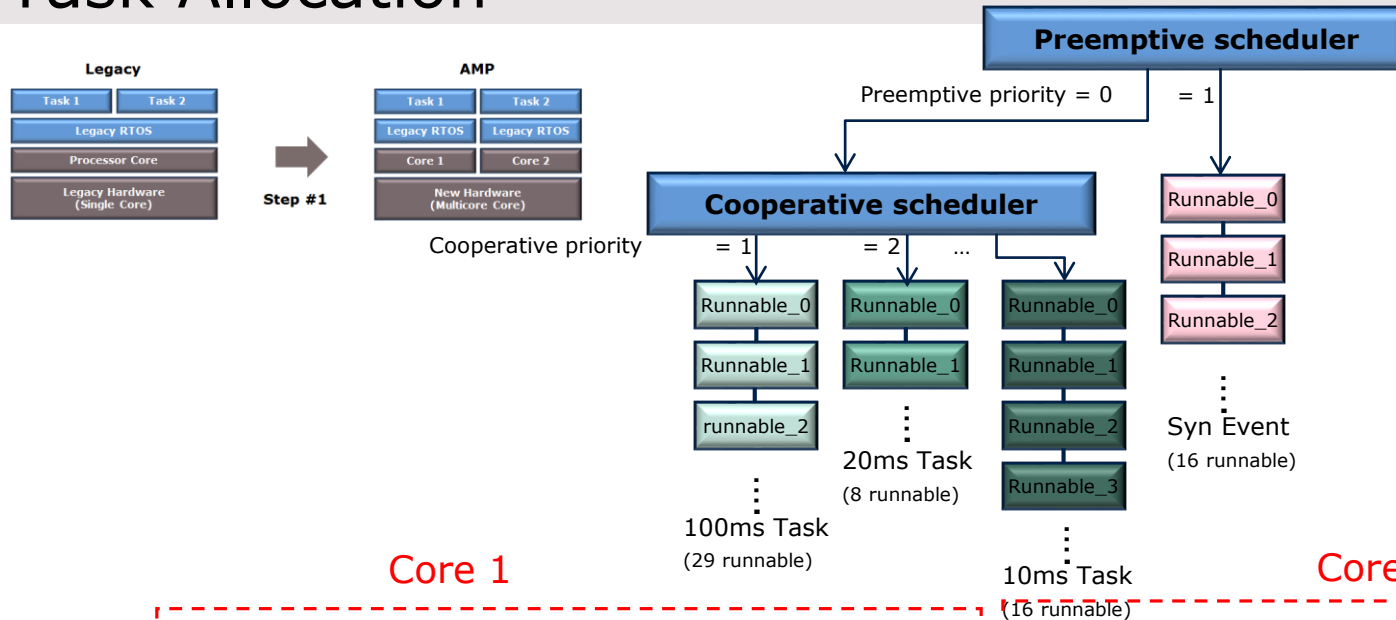
8000 0000 _H - 802F FFFF _H	3 Mbyte	Program Flash 0 (PF0)	Access	SRIBE	Cached PFLASH
8030 0000 _H - 805F FFFF _H	3 Mbyte	Program Flash 1 (PF1)	Access	SRIBE	
8060 0000 _H - 808F FFFF _H	3 Mbyte	Program Flash 2 (PF2)	Access	SRIBE	
8090 0000 _H - 80BF FFFF _H	3 Mbyte	Program Flash 3 (PF3)	Access	SRIBE	
80C0 0000 _H - 80EF FFFF _H	3 Mbyte	Program Flash 4 (PF4)	Access	SRIBE	
80F0 0000 _H - 80FF FFFF _H	1 Mbyte	Program Flash 5 (PF5)	Access	SRIBE	
A000 0000 _H - A02F FFFF _H	3 Mbyte	Program Flash 0 (PF0)	Access	SRIBE	Non-cached PFLASH
A030 0000 _H - A05F FFFF _H	3 Mbyte	Program Flash 1 (PF1)	Access	SRIBE	
A060 0000 _H - A08F FFFF _H	3 Mbyte	Program Flash 2 (PF2)	Access	SRIBE	
A090 0000 _H - A0BF FFFF _H	3 Mbyte	Program Flash 3 (PF3)	Access	SRIBE	
A0C0 0000 _H - A0EF FFFF _H	3 Mbyte	Program Flash 4 (PF4)	Access	SRIBE	
A0F0 0000 _H - A0FF FFFF _H	1 Mbyte	Program Flash 5 (PF5)	Access	SRIBE	
A100 0000 _H - A11F FFFF _H	2 Mbyte	Reserved (for PFLASH)	SRIBE	SRIBE	
AF00 0000 _H - AF0F FFFF _H	1 Mbyte	Data Flash 0 EEPROM (DF0) Host Comd. Sequence Interpreter	Access	Access ⁵⁾	Non-cached DFLASH
AF10 0000 _H - AF3F FFFF _H	3 Mbyte	Reserved	SRIBE	SRIBE	
AF40 0000 _H - AF40 5FFF _H	24 Kbyte	Data Flash 0 UCB (DF0)	Access	SRIBE	
AF40 6000 _H - AF7F FFFF _H	-	Reserved	SRIBE	SRIBE	
AF80 0000 _H - AF80 FFFF _H	64 Kbyte	Data Flash 0 CFS (DF0)	Access	SRIBE	
AF81 0000 _H - AFBF FFFF _H	-	Reserved	SRIBE	SRIBE	
AFC0 0000 _H - AFC1 FFFF _H	128 Kbyte	Data Flash 1 EEPROM (DF1) HSM Comd. Sequence Interpreter	Access	Access ⁶⁾	

AURIX 2G Multi-Core Address Map

DLMU, LMU and DAM

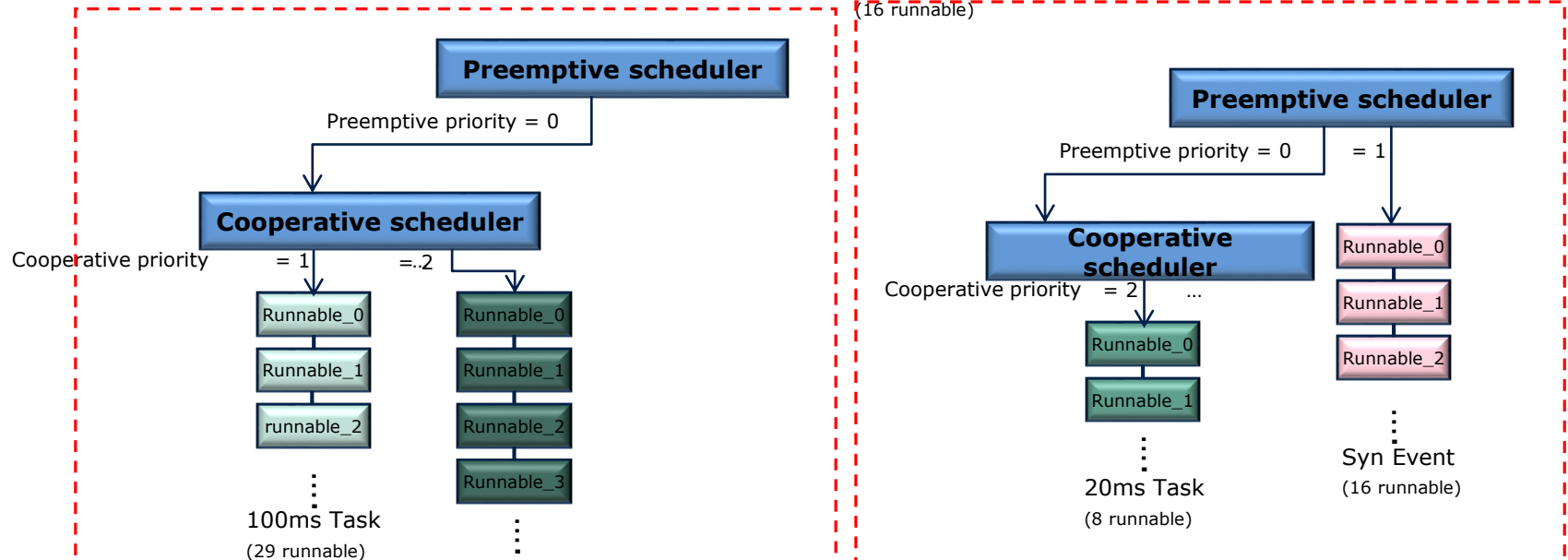
9000 0000 _H - 9000 FFFF _H	64 Kbyte	LMU (CPU0 DLMU)	Access	Access	Cached LMU
9001 0000 _H - 9001 FFFF _H	64 Kbyte	LMU (CPU1 DLMU)	Access	Access	
9002 0000 _H - 9002 FFFF _H	64 Kbyte	LMU (CPU2 DLMU)	Access	Access	
9003 0000 _H - 9003 FFFF _H	64 Kbyte	LMU (CPU3 DLMU)	Access	Access	
9004 0000 _H - 9007 FFFF _H	256 Kbyte	LMU (LMU0 LMURAM)	Access	Access	
9008 0000 _H - 900B FFFF _H	256 Kbyte	LMU (LMU1 LMURAM)	Access	Access	
900C 0000 _H - 900F FFFF _H	256 Kbyte	LMU (LMU2 LMURAM)	Access	Access	
9010 0000 _H - 9010 FFFF _H	64 Kbyte	LMU (CPU4 DLMU)	Access	Access	
9011 0000 _H - 9011 FFFF _H	64 Kbyte	LMU (CPU5 DLMU)	Access	Access	
9012 0000 _H - 903F FFFF _H	-	Reserved	SRIBE	SRIBE	Cached DAM
9040 0000 _H - 9040 7FFF _H	32 Kbyte	DAM (DAM0 RAM0)	Access	Access	
9040 8000 _H - 9040 FFFF _H	32 Kbyte	DAM (DAM0 RAM1)	Access	Access	
9041 0000 _H - 9041 7FFF _H	32 Kbyte	DAM (DAM1 RAM0)	Access	Access	
9041 8000 _H - 9041 FFFF _H	32 Kbyte	DAM (DAM1 RAM1)	Access	Access	
B000 0000 _H - B000 FFFF _H	64 Kbyte	LMU (CPU0 DLMU)	Access	Access	Non-cached LMU
B001 0000 _H - B001 FFFF _H	64 Kbyte	LMU (CPU1 DLMU)	Access	Access	
B002 0000 _H - B002 FFFF _H	64 Kbyte	LMU (CPU2 DLMU)	Access	Access	
B003 0000 _H - B003 FFFF _H	64 Kbyte	LMU (CPU3 DLMU)	Access	Access	
B004 0000 _H - B007 FFFF _H	256 Kbyte	LMU (LMU0 LMURAM)	Access	Access	
B008 0000 _H - B00B FFFF _H	256 Kbyte	LMU (LMU1 LMURAM)	Access	Access	
B00C 0000 _H - B00F FFFF _H	256 Kbyte	LMU (LMU2 LMURAM)	Access	Access	
B010 0000 _H - B010 FFFF _H	64 Kbyte	LMU (CPU4 DLMU)	Access	Access	
B011 0000 _H - B011 FFFF _H	64 Kbyte	LMU (CPU5 DLMU)	Access	Access	
B012 0000 _H - B03F FFFF _H	-	Reserved	SRIBE	SRIBE	Non-cached DAM
B040 0000 _H - B040 7FFF _H	32 Kbyte	DAM (DAM0 RAM0)	Access	Access	
B040 8000 _H - B040 FFFF _H	32 Kbyte	DAM (DAM0 RAM1)	Access	Access	
B041 0000 _H - B041 7FFF _H	32 Kbyte	DAM (DAM1 RAM0)	Access	Access	
B041 8000 _H - B041 FFFF _H	32 Kbyte	DAM (DAM1 RAM1)	Access	Access	

Multi-core Code and Data allocation Task Allocation



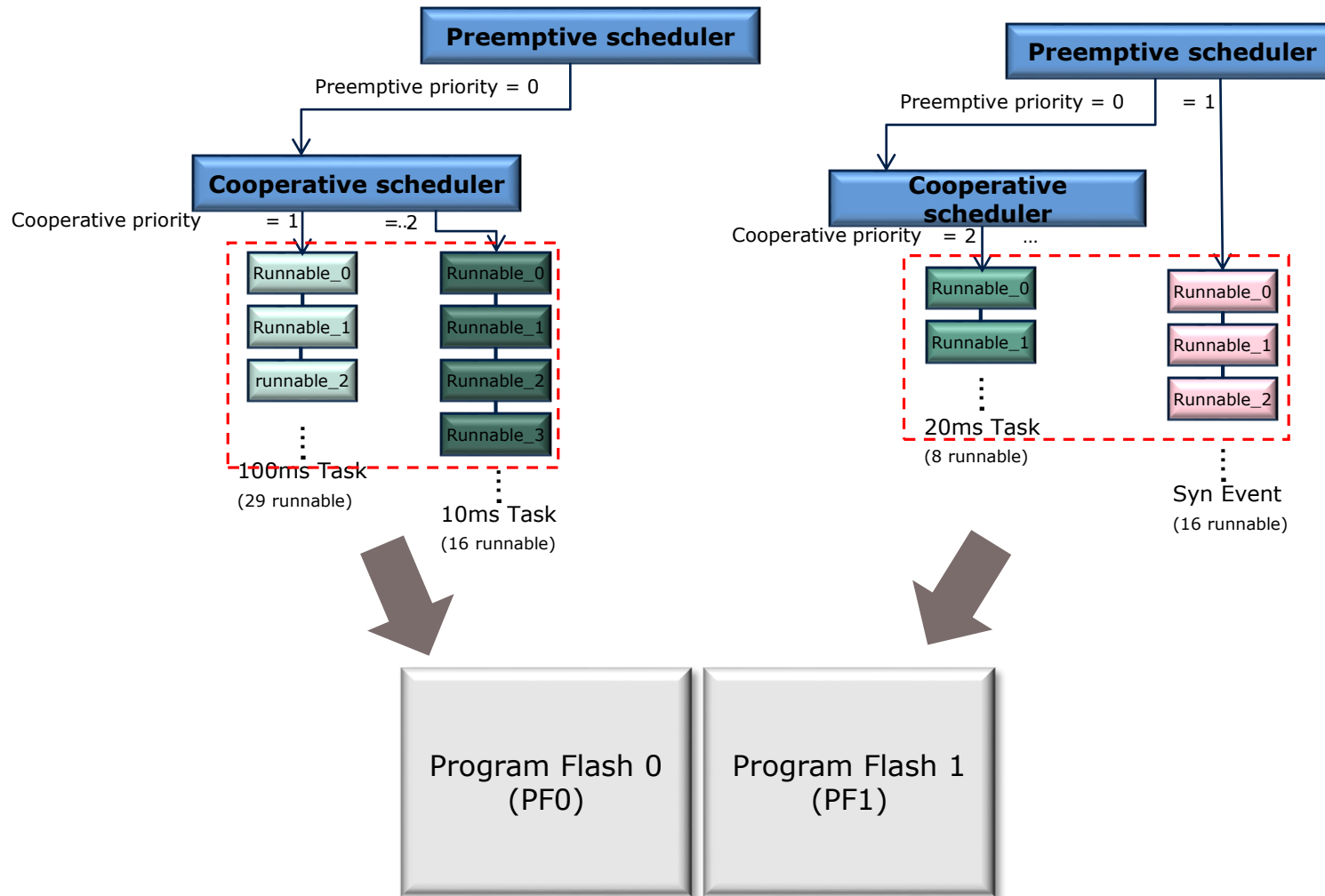
Core 1

Core 0



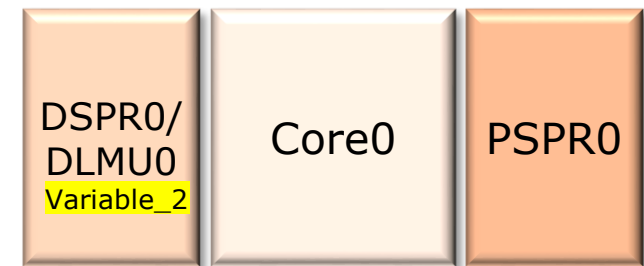
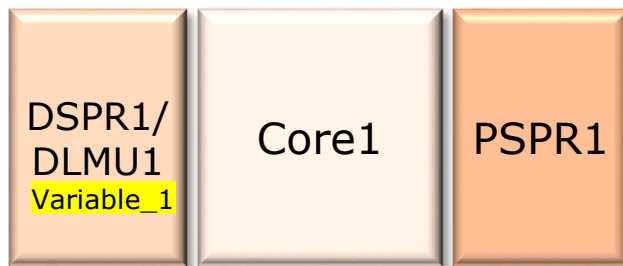
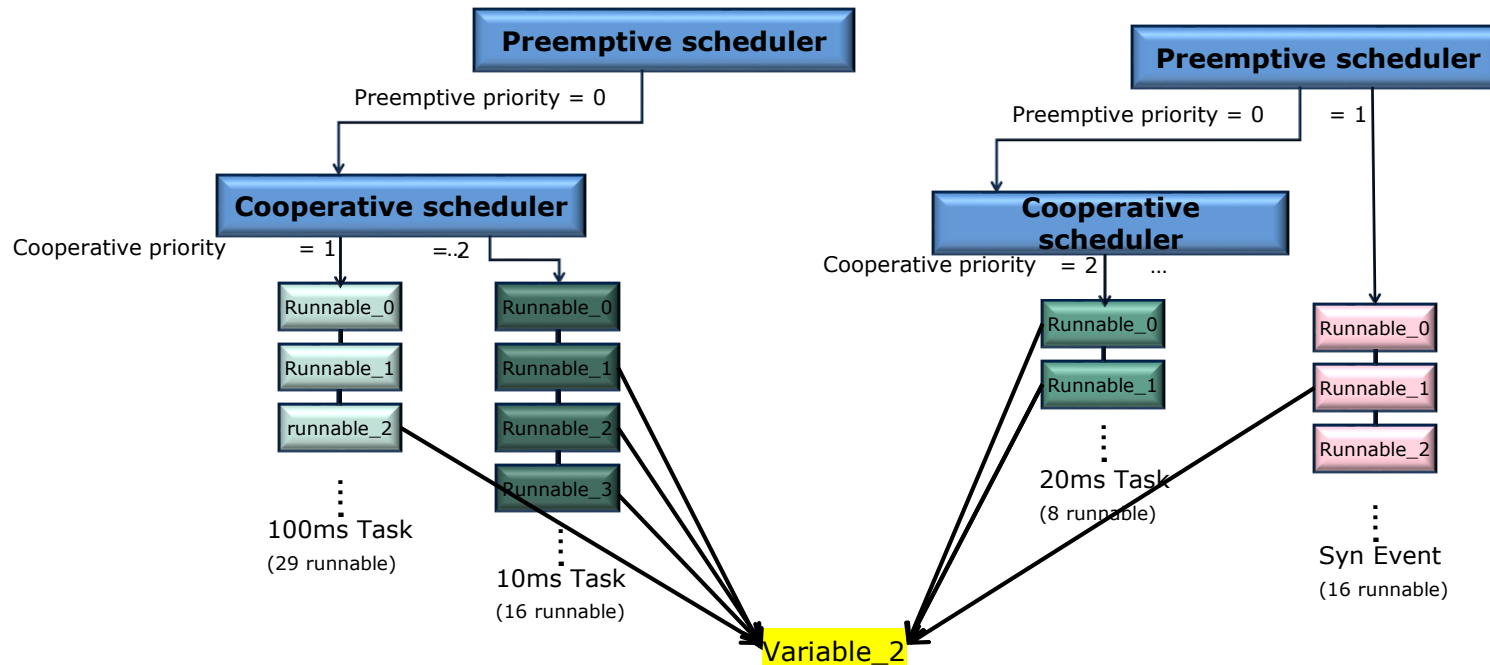
Multi-core Code and Data allocation

Program Memory Allocation



Multi-core Code and Data allocation

Data Memory Allocation

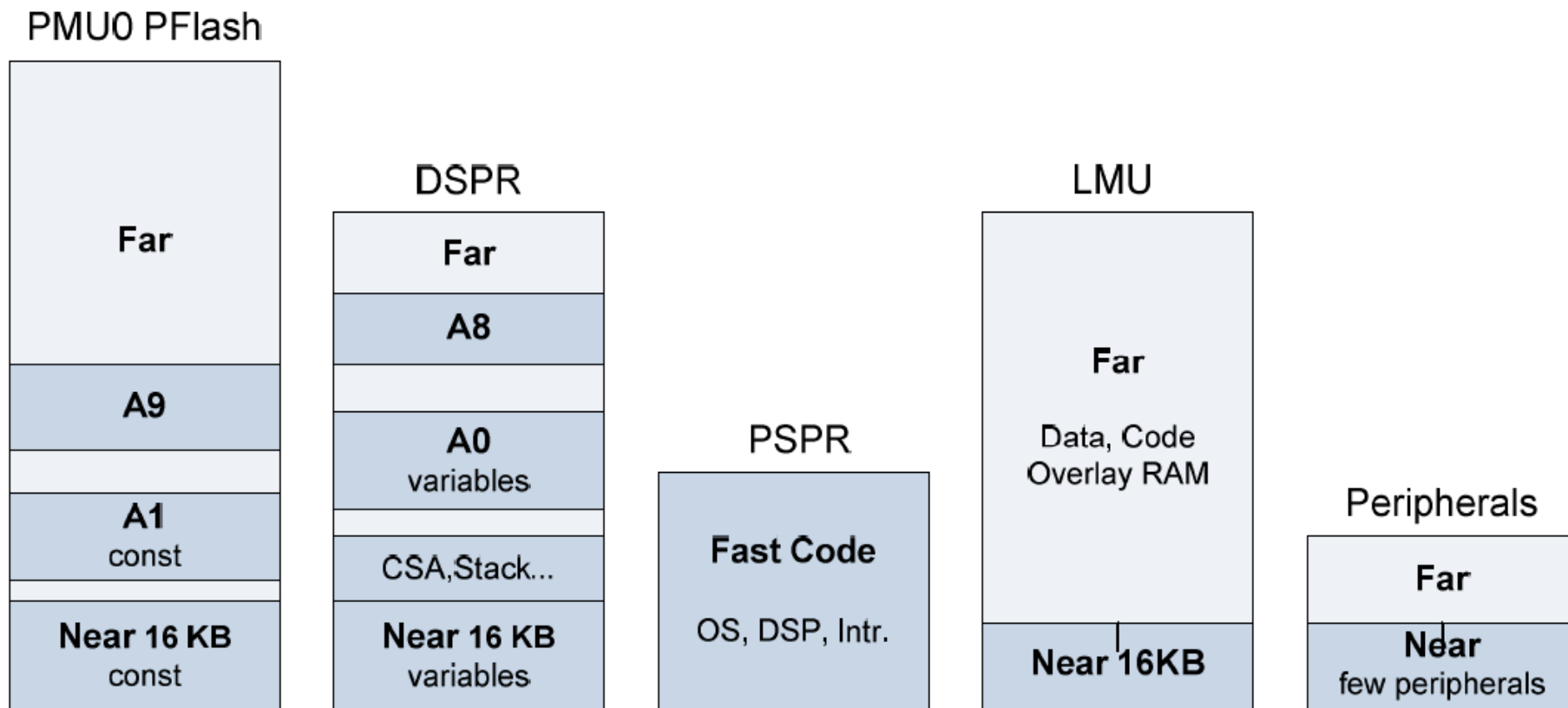


Multi-core Code and Data allocation

Some application Hint for code and data assignment

- › Interrupt vector table : Core local flash bank
- › Trap vector table : Core local flash bank
- › Context Save Area : Core local DSPR
- › User stack : Core local DSPR
- › Interrupt stack : Core local DSPR
- › Private variable : Core local DSPR and DLUM
- › Private constant : Core local flash bank
- › Private function : Core local PSPR, local flash bank
- › Public variable :
 - Non-cached access mode,
 - 32 bit alignment,
 - Spin-Lock protection when width >32 bit
 - data buffer
 - Location : DSPR, DLMUx, LMU or DAM (time constraint and access efficient)
- › Public function : copy function to local PSPR

Memory Location



Agenda

- 1 AURIX 2G Family and Multi-core Architecture
- 2 Memory Map, Code and Data allocation
- 3 **Associate Core, Data and Code in Tasking**
- 4 Associate Core, Data and Code in GNU
- 5 Some common issues in Multi-core system

Section types and Address spaces

Section type	Name prefix	Description
code	.text	program code
neardata	.zdata	initialized __near data
fardata	.data	initialized __far data
nearrom	.zrodata	constant __near data
farrom	.rodata	constant __far data
nearbss	.zbss	uninitialized __near data (cleared)
farbss	.bss	uninitialized __far data (cleared)
nearnoclear	.zbss	uninitialized __near data
farnoclear	.bss	uninitialized __far data
a0data	.data_a0	initialized __a0 data
a0rom	.rodata_a0	constant __a0 data
a0bss	.bss_a0	uninitialized __a0 data (cleared)

Section type	Name prefix	Description
a1data	.data_a1	initialized __a1 data
a1rom	.rodata_a1	constant __a1 data
a1bss	.bss_a1	uninitialized __a1 data (cleared)
a8data	.data_a8	initialized __a8 data
a8rom	.rodata_a8	constant __a8 data
a8bss	.bss_a8	uninitialized __a8 data (cleared)
a9data	.data_a9	initialized __a9 data
a9rom	.rodata_a9	constant __a9 data
a9bss	.bss_a9	uninitialized __a9 data (cleared)

Space	Id	MAU	Description	ELF sections
linear	1	8	Linear address space	.text*, .data*, .sdata*, .ldata*, .rodata*, .bss*, .sbss*, table, istack, ustack
abs24	2	8	Absolute 24-bit addressable space	
abs18	3	8	Absolute 18-bit addressable space	.zdata, .zrodata, .zbss
csa	4	8	Context Save Area	csa.*

Compiler memory qualifiers

Qualifiers	Description	Location	Maximum object size	Pointer size	Section types
<code>__near</code>	Near data, direct addressable	First 16 kB of a 256 MB block	16 kB	32-bit	neardata, nearrom, nearbss, nearnoclear
<code>__far</code>	Far data, indirect addressable	Anywhere	no limit	32-bit	fardata, farrom, farbss, farnoclear
<code>__a0</code>	Small data	Sign-extended 16-bit offset from address register A0	64 kB	32-bit	a0data, a0bss
<code>__a1</code>	Literal data, read-only	Sign-extended 16-bit offset from address register A1	64 kB	32-bit	a1rom
<code>__a8</code>	Data, reserved for OS	Sign-extended 16-bit offset from address register A8	64 kB	32-bit	a8data, a8rom, a8bss
<code>__a9</code>	Data, reserved for OS	Sign-extended 16-bit offset from address register A9	64 kB	32-bit	a9data, a9rom, a9bss

Data and Core Association

Data core association	Memory qualifier	Accessible from [*]	Number of instances	Allocation in
Share	<code>__share</code>	All cores	One instance. The data object is shared between cores.	Global RAM or core-local RAM Note: when allocated in core-local ram the shared object is accessed via the mirror page.
Private	<code>__private0</code> <code>__private1</code> <code>__private2</code>	One core	One instance. For one specific core.	Core-local RAM
Clone	<code>__clone</code>	All cores	Multiple instances. For each core one instance is allocated. All instances will be located at identical addresses.	Core-local RAM

Data and Core Association

Local Memory for Core 0 (1)

› Core 0 DSPR address

```
memory dspr0 // Data Scratch Pad Ram
{
    mau = 8;
    size = 112k;
    type = ram;
    map (dest=bus:tc0:fpi_bus, dest_offset=0xd0000000, size=112k, priority=1, exec_priority=0);
    map (dest=bus:sri, dest_offset=0x70000000, size=112k);
}
```

› Core 0 PSPR address

```
memory pspr0 // Program Scratch Pad Ram CPU0
{
    mau = 8;
    size = 64k;
    type = ram;
    map (dest=bus:tc0:fpi_bus, dest_offset=0xc0000000, size=64k, exec_priority=1);
    map (dest=bus:sri, dest_offset=0x70100000, size=64k);
}
```

Data and Core Association

Local Memory for Core 0 (1)

› Core 0 LPB address

```
memory pflash0 // Program Flash CPU0
{
    mau = 8;
    size = 3M;
    type = rom;
    map      cached (dest=bus:sri, dest_offset=0x80000000,      size=3M);
    map not_cached (dest=bus:sri, dest_offset=0xa0000000, reserved, size=3M);
}
```

› Core 0 DLMU address

```
memory dlmucpu0 // DLMU CPU0
{
    mau = 8;
    size = 64k;
    type = ram;
    map      cached (dest=bus:sri, dest_offset=0x90000000,      size=64k);
    map not_cached (dest=bus:sri, dest_offset=0xb0000000, reserved, size=64k);
}
```

Data and Core Association

Private global variable

- How to define a private variable for core 0 in **Core 0 Local DSPR** and **Public address**
`__private0 unsigned int test ;`

Chip	Group	Section	Size (MAU)	Space addr	Chip addr	Alignment
mpe:dspr0		.bss.private0.TC275_Ram_Usage.test (167)	0x00000004	0x70000000	0x0	0x00000002
mpe:dspr0		.bss.private0.TC275_Ram_Usage.test1 (168)	0x00000004	0x70000004	0x00000004	0x00000002
mpe:dspr0		istack_tc0 (352)	0x00000400	0x70000008	0x00000008	0x00000008
mpe:dspr0		ustack_tc0 (351)	0x00004000	0x70005000	0x00005000	0x00000008

- How to relocate a private variable for core 0 to **private address**
`section_layout mpe:tc0:linear (direction = low_to_high)`
`{`
`group Core0_Data_RAM (run_addr = [0xD0000000..0xD00000FF], ordered)`
`{`
`select ".bss.private0*";`
`}`
`}`

Chip	Group	Section	Size (MAU)	Space addr	Chip addr	Alignment
mpe:dspr0		.bss.private0 (168)	0x00000004	0x70000004	0x00000004	0x00000002
mpe:dspr0		istack_tc0 (352)	0x00000400	0x70000008	0x00000008	0x00000008
mpe:dspr0		ustack_tc0 (351)	0x00004000	0x70005000	0x00005000	0x00000008
mpe:dspr0	Core0_Data_RAM	.bss.private0.Core0 (167)	0x00000004	0xd0000000	0x0	0x00000002

Data and Core Association

Share global variable (1)

- › How to define a share variable?
`__share unsigned int test ; // unsigned int test;`

0x10000000	test	mpe:vtc:abs18
0x70000400	_lc_ue_istack	mpe:tc0:linear

- › How to define a share variable in **Core 0 DLMU0**?

```
#pragma section fardata core0_dlmu
__far int dlm_u_core0;
#pragma section fardata restore
group core0dlmu (run_addr=mem:mpe:d1mucpu0)
{
    select "*.core0_dlm_u";
}
```

0x90000000	dlmu_core0	mpe:vtc:linear
0x90000000	_lc_gb_dlm_u0	mpe:vtc:linear
0x90000004	_lc_ge_dlm_u0	mpe:vtc:linear

- › How to define a share variable in **Core 0 DSPR**?

```
#pragma section fardata core0_dspr
__far int pspr_core0 =0;
#pragma section fardata restore
group core0dspr (run_addr=mem:mpe:dspr0)
{
    select "*.core0_dspr";
}
```

0x70000000	pspr_core0	mpe:vtc:linear
0x70000000	_lc_gb_core0dspr	mpe:vtc:linear
0x70000004	_lc_ge_core0dspr	mpe:vtc:linear

- › Q : How to define a share variable in **Imuram0 and dam0ram0**

Data and Core Association

Clone global variable

- › Define a clone variable for all core

__clone int clone_p0_i;

+ Space mpe:vtc:mpe_tc0_abs18|mpe_tc1_abs18|mpe_tc2_abs18 (MAU = 8bit)

Chip	Group	Section	Size (MAU)	Space addr	Chip addr	Alignment
mpe:dspr0+mpe:dspr1+mpe:dspr2		.zbss.clone.core0_main.clone_p0_i (7)	0x00000004	0xd0000000	0x0	0x00000002

Code and Core Association

Code core association	Memory qualifier	Executes on [*]	Number of instances	Allowed access of DCA qualified data	Allocation in
Share	<code>__share</code>	Any core	One instance. The code is shared between cores.	Shared Cloned (of executing core)	PFLASH_0, PFLASH_1 or core-local RAM
Private	<code>__private0</code> <code>__private1</code> <code>__private2</code>	One core only	One instance.	Shared Private Cloned	Core-local RAM
Clone	<code>__clone</code>	Any core	Multiple instances. Each code instance is executed by one core.	Shared Cloned	Core-local RAM

Code and Core Association

Private function

- How to define a private function for core 0 in **Core 0 Local PSPR** (copy code from flash to local PSPR)

```
__private0 void core0_main(void)
```

Chip	Group	Section	Size (MAU)	Space addr
mpe:dspr0		istack_tc0 (1033)	0x00000400	0x70000008
mpe:dspr0		ustack_tc0 (1032)	0x00004000	0x70005000
mpe:pspr0		.text.private0.core0_main.core0_main (4)	0x0000016a	0x70100260
mpe:pspr0		.text.private0.core0_main.private_0_code (2)	0x0000000c	0x701003ca

Chip	Group	Section	Size (MAU)	Space addr
mpe:dspr0		.bss.cstart._tcx_end_c_init (85)	0x00000004	0x70000000
mpe:pflash0		[.text.private0.core0_main.core0_main] (940)	0x0000016a	0x80000030

- How to relocate a private function for core 0 to the **private address** section_layout mpe:tc0:linear (direction = low_to_high)

```
{
    group Core0_Code_RAM ( run_addr = [0xC0000000..0xc00000FF], ordered){
        select ".text.private0.*.core0_main";
    }
}

+ Space mpe:tc0:linear (MAU = 8bit)
```

Chip	Group	Section	Size (MAU)	Space addr	Chip addr	Alignment
mpe:dspr0		istack_tc0 (518)	0x00000400	0x70000008	0x00000008	0x00000008
mpe:dspr0		ustack_tc0 (517)	0x00004000	0x70005000	0x00005000	0x00000008
mpe:pspr0	Core0_Code_RAM	.text.private0.core0_main.core0_main (6)	0x0000005a	0xc0000000	0x0	0x00000002
mpe:pspr0	Core0_Code_RAM	.text.private0.core0_main.private_0_code (5)	0x0000000c	0xc000005a	0x0000005a	0x00000002

› Define a shared function in **Core 1 Local Pflash**

```
#pragma section code pflash1
void wait_forever(void)
{
    _Pragma("warning 557");
    while (1);
    _Pragma("warning restore");
}
#pragma section code restore
```

```
group pf1 (run_addr=mem:mpe:pflash1)
{
    select "*.pflash1";
}
```

0x80300000	_lc_gb_pf1
0x80300000	wait_forever
0x80300002	_lc_ge_pf1

› Define a shared function in **Core 1 Local PSPR**

```
#pragma section code core1_pspr
void wait_forever(void)
{
    _Pragma("warning 557");
    while (1);
    _Pragma("warning restore");
}
#pragma section code restore
```

```
group pspr1
(run_addr=mem:mpe:pspr1, ordered, copy)
{
    select "*.core1_pspr";
}
```

0x60100000	wait_forever	mpe:vtc:linear
0x60100000	_lc_gb_pspr1	mpe:vtc:linear

› Q : what is the flash address for the shared function

Code and Core Association

Clone global function

- › Define a clone function for all core(it will be copied from flash to separate PSPR)

`__clone void IfxCpuSyncDemo_run(unsigned char core_id)`

Chip	Group	Section	Size (MAU)	Space addr
mpe:pspr0+mpe:pspr1+mpe:pspr2+mpe:pspr3+mpe:pspr4+mpe:pspr5		.text.clone.main.IfxCpuSyncDemo_run (381)	0x00000254	0xc0000000
mpe:pspr0+mpe:pspr1+mpe:pspr2+mpe:pspr3+mpe:pspr4+mpe:pspr5		.text.clone.core0_main.clone_code (3)	0x0000000c	0xc0000254

Agenda

- 1 AURIX 2G Family and Multi-core Architecture
- 2 Memory Map, Code and Data allocation
- 3 Associate Core, Data and Code in Tasking
- 4 **Associate Core, Data and Code in GNU**
- 5 Some common issues in Multi-core system

Data and Core Association

Define a variable at local DLMU&DSPR



› Section define(Lcf_Gnuc.lsl)

```
/*DLMU1 Sections*/  
CORE_SEC(.lmudata) : FLAGS(awl)  
{  
    *(.lmudata_cpu1)  
} > dlmu1 AT> pfls0
```

› In the application code

```
volatile uint32 my_var  
__attribute__((section(".lmudata_cpu1")));
```

› Section define(Lcf_Gnuc.lsl)

```
/*DSPR1 Sections*/  
CORE_SEC(.dsramdata) : FLAGS(awl)  
{  
    *(.dsramdata_cpu1)  
} > dsram1 AT> pfls0
```

› In the application code :

```
volatile uint32 my_var  
__attribute__((section(".dsramdata_cpu1")));
```

Code and Core Association

Define a function in local PFLASH

- › Define ``.text_cpu1`` in `*.lsl` file,

```
CORE_SEC(.text) : FLAGS(axl)
{
    . = ALIGN(2);
    *Ifx_Ssw_Tcl.* (.text)
    *Ifx_Ssw_Tcl.* (.text.*)
    *Cpul_Main.* (.text)
    *Cpul_Main.* (.text.*)
    *(.text_cpu1)
} > flash1
```

- › Define user function in the source code as following:

#pragma section `".text_cpu1"` axw

```
void IfxApp_Blink_Leds_Init(void) {
    /* Assemble the final configuration structure: */
    const IfxPort_Io_Config conf = {
        ledPinCount,
        (IfxPort_Io_ConfigPin *)ledConfigPin
    };
    /* Call the initialisation function: */
    IfxPort_Io_initModule(&conf);
}
```

#pragma section

Code and Core Association

Allocate and execute a function from local PSRR

- › Define ``.cpu0_psram`` in `*.lsl` file

```
CORE_SEC(.psram_text) : FLAGS(awx)
{
    . = ALIGN(2);
    *(.psram_text_cpu0)
    *(.cpu0_psram)
} > psram0 AT> pfls0
```

- › Define user function in the source code as following:

#pragma section `".cpu0_psram"` axw

```
void IfxApp_Blink_Leds_Init(void) {
    /* Assemble the final configuration structure: */
    const IfxPort_Io_Config conf = {
        ledPinCount,
        (IfxPort_Io_ConfigPin *)ledConfigPin
    };
    /* Call the initialisation function: */
    IfxPort_Io_initModule(&conf);
}
```

#pragma section

```
1445 0x70015c00 0x70015c00 0 g _CSA0
1446 0x70017c00 0x70017c00 0 g _CSA0_END
1447 0x70100000 0x70100015 22 g IfxApp_Blink_Leds_Init
1448 0x80000000 0x80000000 0 g RESET
1449 0x80000000 0x80000000 10 g _START
```


Prelinking with core assignment

- › For assigning an object to the core CPU<N> (where <N> is the index of the core), do the below section define:

```
SECTIONS
```

```
{  
    CORE SEC(.text) 0:  
    {  
        *(.text) *(.text.*)  
    }  
    CORE SEC(.data) 0:  
    {  
        *(.data)  
    }  
}
```

```
tricore-ld -r --core=CPU<N> -T<linker>.ld
```

Core assignment in the source code

- › The following example assign the .text.main section to the CPU0 core by setting the **c0** flag.

```
#pragma section ".text.main" axc0 2
    static void main_core 0 sub(void);
    void main_core0(void) { ... }
#pragma section
```

Locating

For locating multi-core objects, the following core extensions can be used in the linker script:

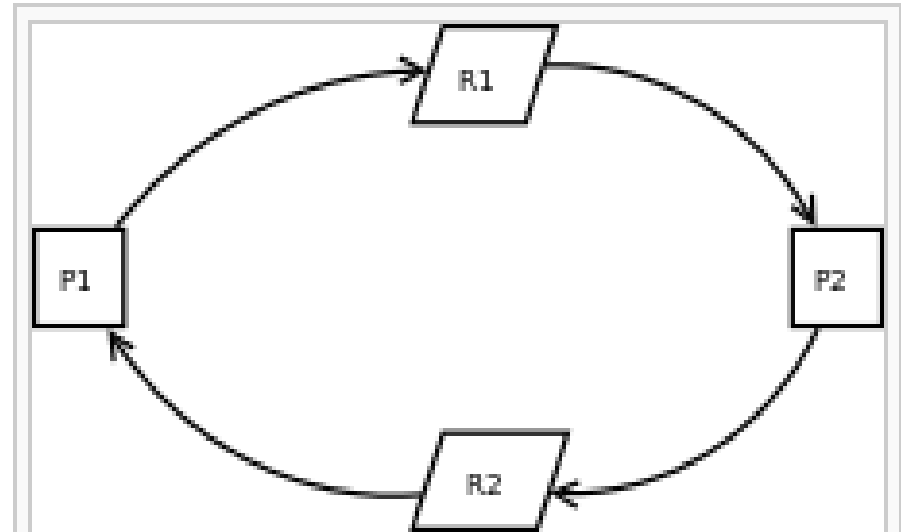
- › `CORE_SEC(section)` :
 - Add a prefix to the section name which includes the actual core number, when `CORE_ID` is not `GLOBAL`. E.g. `CORE_SEC(.output)` will get the name `.CPU<N>.output`
- › `CORE_SYM(symbol)`
 - Add a suffix to the symbol name which includes the actual core number. E.g. `CORE_SYM(symbol)` will get the name `symbol_CPU<N>_`
- › `CORE_ID=<label>`
 - The built-in symbol `CORE_ID` will set the actual core number used in the interpretation of the following linker script commands. `<label>` can be `GLOBAL` or between `CPU0 ... CPU6`.
- › `CORE_ALIAS`
 - A user-defined prefix of core output sections can be set. E.g. `CORE_ALIAS ("MASTER", CPU0)`

Agenda

- 1 AURIX 2G Family and Multi-core Architecture
- 2 Memory Map, Code and Data allocation
- 3 Associate Core, Data and Code in Tasking
- 4 Associate Core, Data and Code in GNU
- 5 **Some common issues in Multi-core system**

Deadlock

- › Link :
<https://en.wikipedia.org/wiki/Deadlock>
- › Condition for Deadlock
 - Mutual exclusion
 - Hold and wait or resource holding
 - No preemption
 - Circular wait
- › Avoiding database deadlock:
 - An effective way to avoid database deadlocks is to follow this approach from the *Oracle Locking Survival Guide*:



Both processes need resources to continue execution. *P1* requires additional resource *R1* and is in possession of resource *R2*, *P2* requires additional resource *R2* and is in possession of *R1*; neither process can continue.

Race condition

- › Link :
https://en.wikipedia.org/wiki/Race_condition
- › Condition for Deadlock
 - Access simultaneously
 - Without locking or synchronization
- › Solution for Race condition :
 - Mutually exclusive

- › two threads run successively.

Thread 1	Thread 2		Integer value
			0
read value		←	0
increase value			0
write back		→	1
	read value	←	1
	increase value		1
	write back	→	2

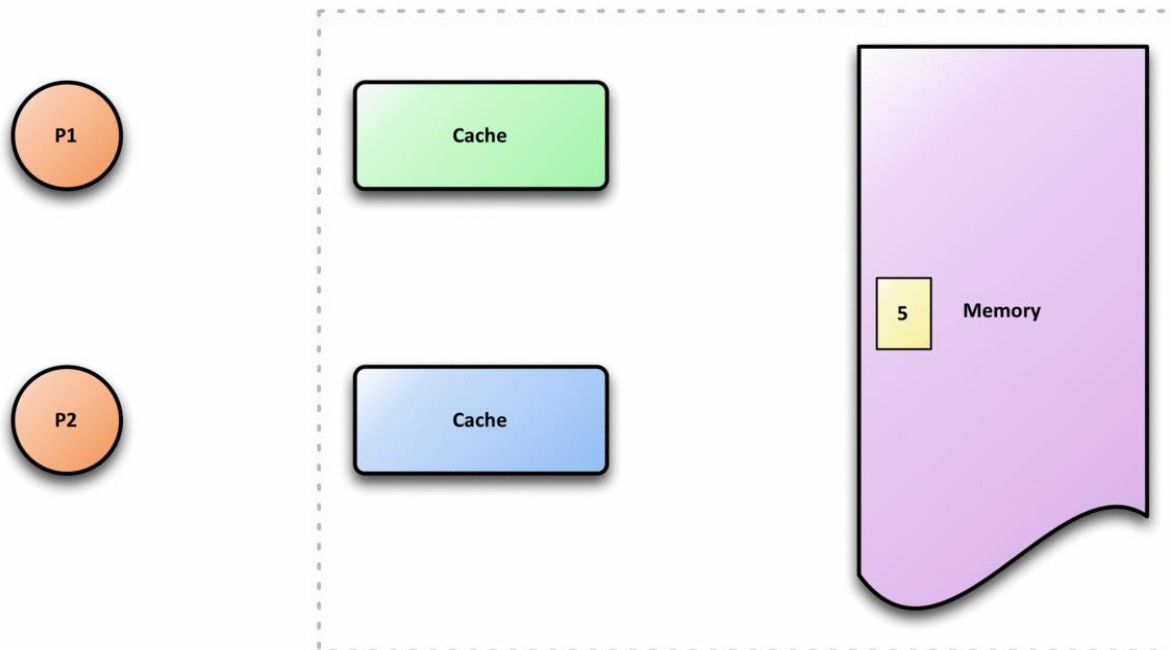
- › two threads run simultaneously without locking or synchronization

Thread 1	Thread 2		Integer value
			0
read value		←	0
	read value	←	0
increase value			0
	increase value		0
write back		→	1
	write back	→	1

Priority Inversion

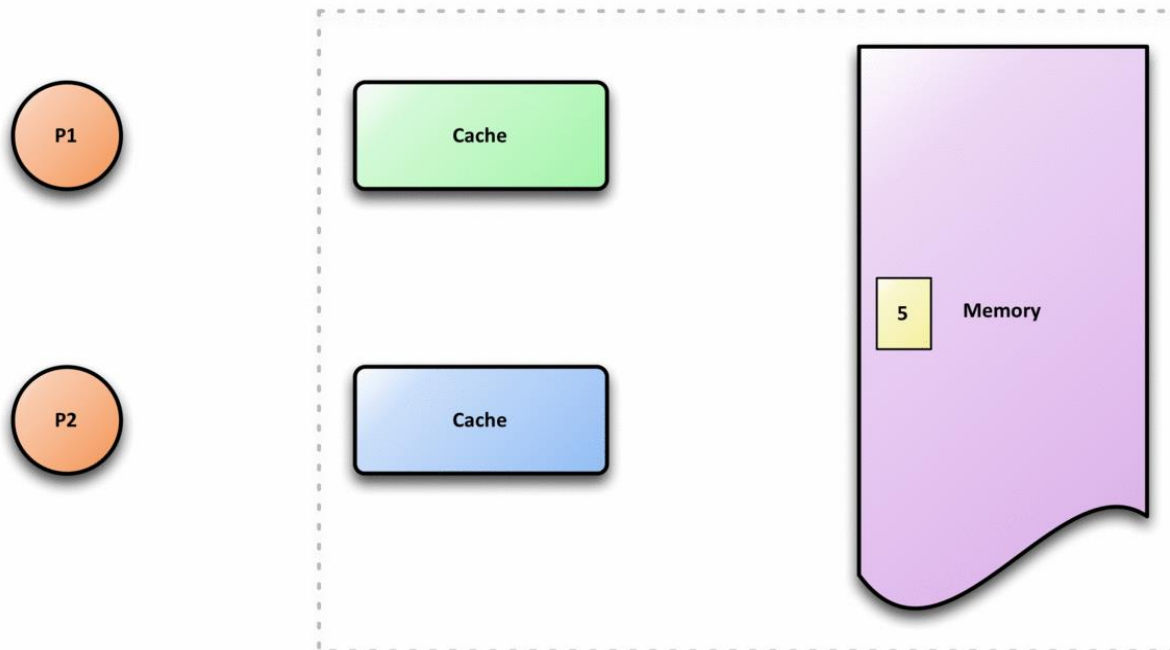
- › Link : https://en.wikipedia.org/wiki/Priority_inversion
- › Explain : In computer science, priority inversion is a problematic scenario in scheduling in which a high priority task is indirectly preempted by a medium priority task effectively "inverting" the relative priorities of the two tasks. “所谓优先级翻转问题(priority inversion)
- › Solution for Priority
 - Priority ceiling
 - priority inheritance

- › Link : https://en.wikipedia.org/wiki/Cache_coherence
- › Explain : In a shared memory multiprocessor system with a separate cache memory for each processor, it is possible to have many copies of shared data: one copy in the main memory and one in the local cache of each processor that requested it.



No cache coherence in AURIX!!!

- › Link : https://en.wikipedia.org/wiki/Cache_coherence
- › Explain : In a shared memory multiprocessor system with a separate cache memory for each processor, it is possible to have many copies of shared data: one copy in the main memory and one in the local cache of each processor that requested it.



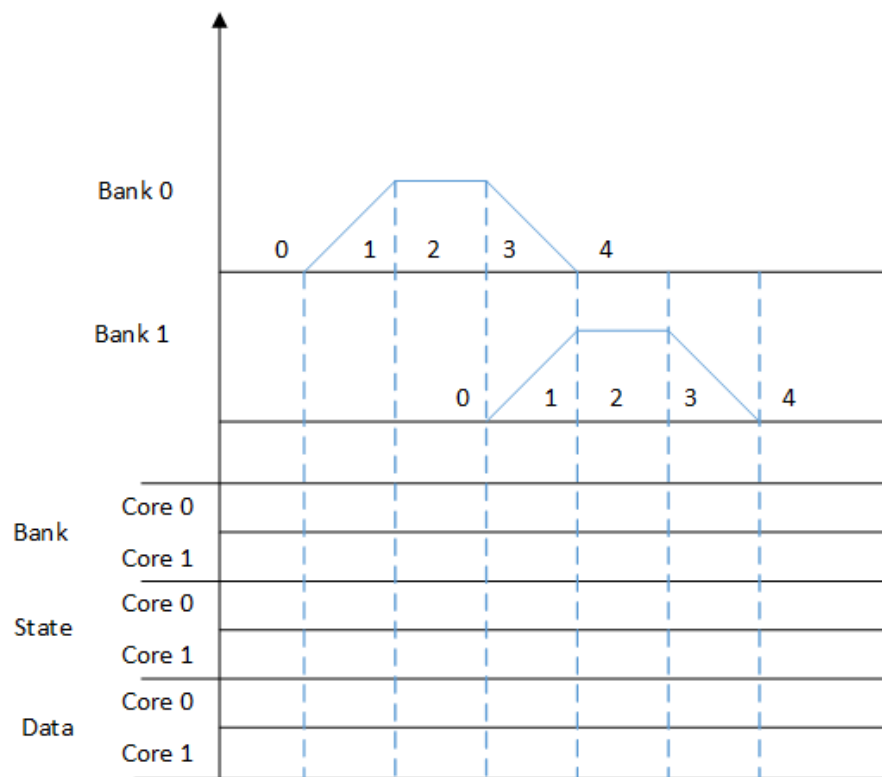
No cache coherence in AURIX!!!

Application Hints for Data coherence (1)

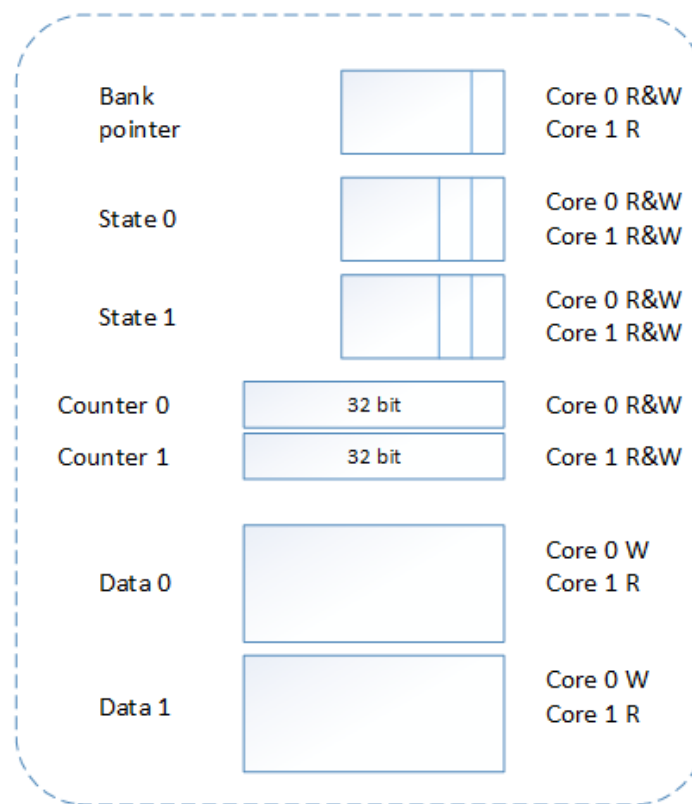
- › Non-cache access
 - LMU/DLMU/DAM (DCON0.DCBYP, PMA0)
 - 0x9xxxxxxx : Cached access
 - 0xBxxxxxxx : Non-cache access
- › 32 bit alignment
- › Atomic operation
 - SWAP.W //Swap with Data Register
 - LDMST //Load-Modify-Store
 - ST.T //Store Bit
 - SWAPMSK //swap Mask
 - CMPSWAP //Compare and Swap
- › Spin-lock or Mutex

Application Hints for Data coherence (2)

› Data buffer



State = 0 : data block can be written by Core 0
 = 1 : data block is being written by Core 0
 = 2 : data block can be read by Core 1
 = 3 : data block is being reading by Core 1





Part of your life. Part of tomorrow.

