NUS
National University
of Singapore

Address Decoding & DMA

Ravi Suppiah
Lecturer, NUS SoC

Address Decoders

NUS National University of Singapore

- Microprocessor systems often have memory components that are smaller than the addressable memory space.

- Different portions of memory are used for different purposes: RAM, ROM, Memory-Mapped Peripherals
- For a given valid address, only one component must be accessed.

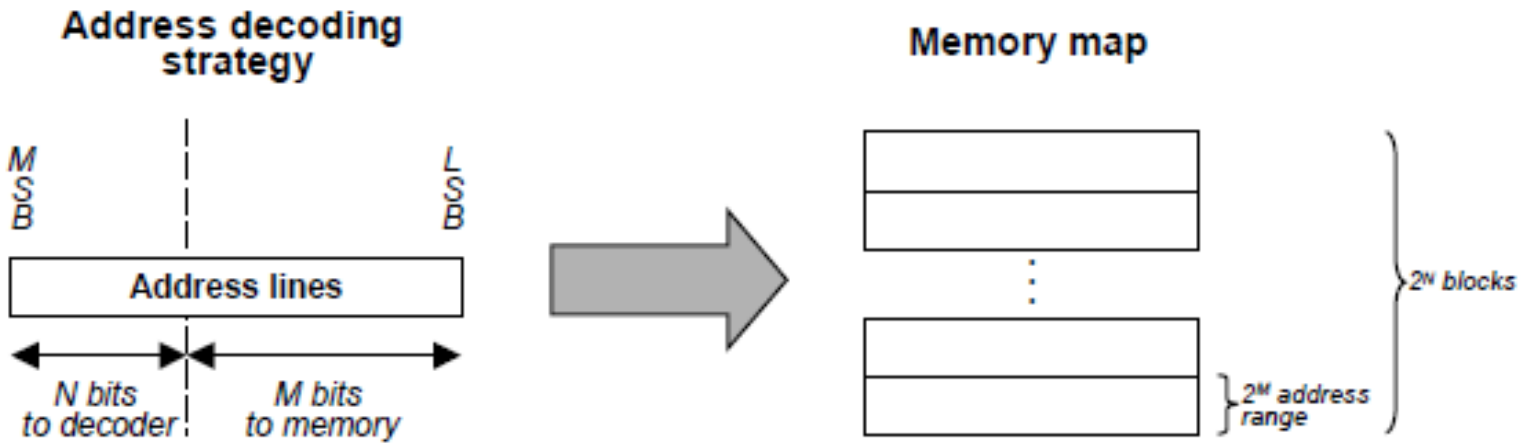
- Address Capacity of Memory
  - Defined by the width of the address bus.
  - If the address bus is 16bits wide, then the microprocessor can address 2^{16} different memory locations.
  - Address Capacity is doubled for each address line that is added.

ARM

Address Decoders

- Address Decoding is the process of generating Chip Select ($\overline{CS}$) signals from the address bus for each device in the system.

- The address bus lines are split into two sections
  - The N most significant bits are used to generate the $\overline{CS}$ signals for the different devices
  - The M least significant signals are passed to the devices as addresses to the different memory cells or internal register.

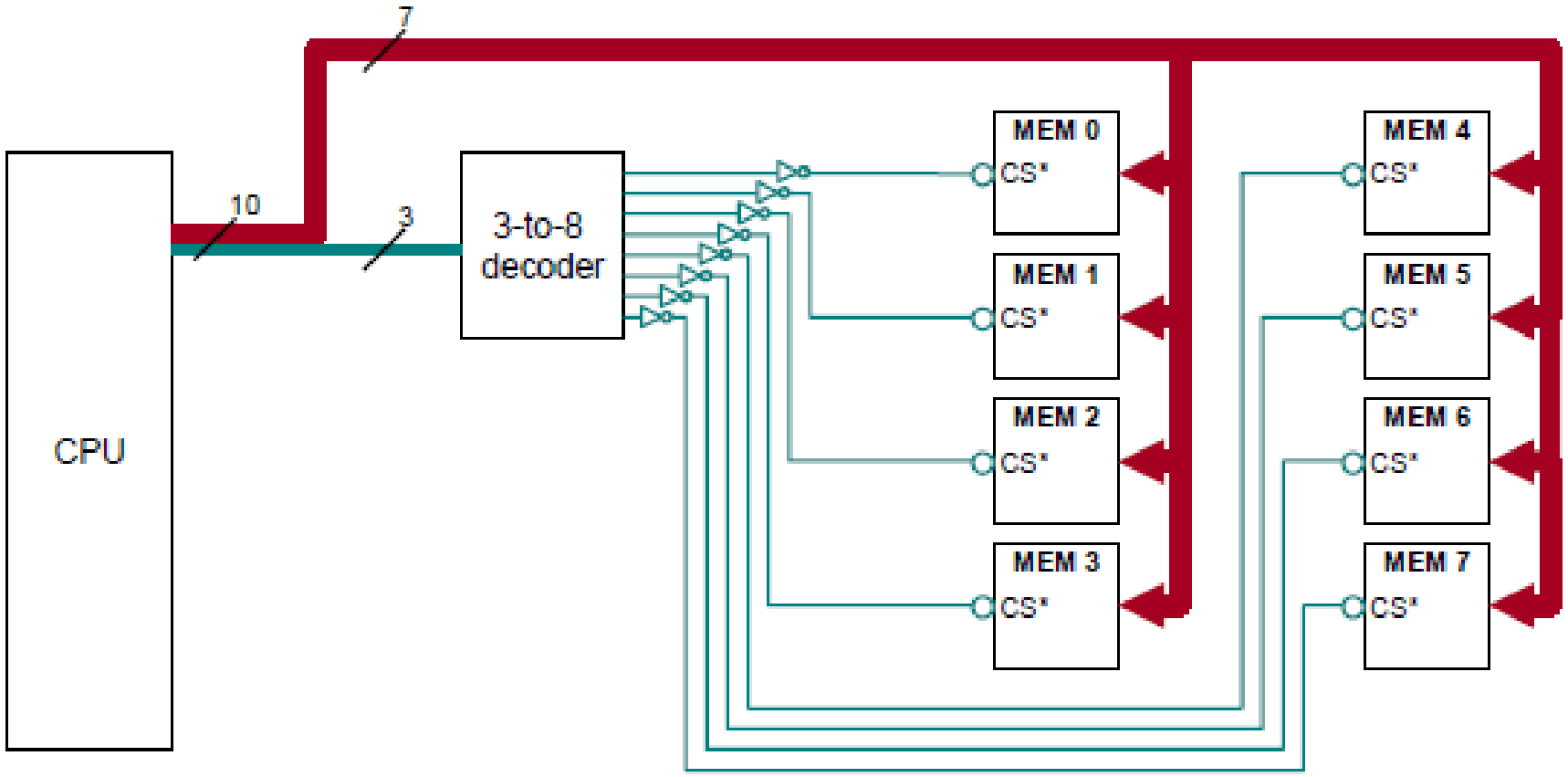


ARM

Address Decoding Example

- A microprocessor has 10 address lines. Its addressable space is 2^{10} => 1KB of memory.
- We wish to populate its entire memory space with 128x8 memory chips.
- SOLUTION
  - We need 8 memory chips => 8 x 128 = 1024
  - We will need 3 address lines to select each one of the 8 chips.
  - Each chip will need 7 address lines to address its internal memory cells.

Address Decoding Example



Memory map

MEM 0
MEM 1
MEM 2
MEM 3
MEM 4
MEM 5
MEM 6
MEM 7

ARM

Address Decoding Methods

- In the previous example we populated the entire address space.
- If only a portion of the addressable space is going to used, there are 2 basic decoding strategies.
- Full Address Decoding
  - All the address lines are used to specify a memory location
  - Each physical memory location is identified by a unique address.

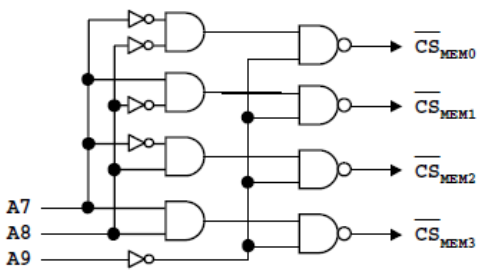
- Partial Address Decoding
  - Only a subset of address lines are needed to point to the physical memory locations.
  - Each physical memory location is identified by several possible addresses.

Full Address Decoding

- Assuming the same microprocessor with 10 address lines (1KB memory space)
  - Now we only want 512 bytes of memory.
  - We must still use 128-Byte memory chips.
  - Place the Physical Memory on the upper half of the memory map.

Used for Address Decoding — *Used to reference memory cells on each memory IC*

Device	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
MEM 0	0	0	0	X	X	X	X	X	X	X
MEM 1	0	0	1	X	X	X	X	X	X	X
MEM 2	0	1	0	X	X	X	X	X	X	X
MEM 3	0	1	1	X	X	X	X	X	X	X



Memory map

MEM 0
MEM 1
MEM 2
MEM 3
Not used
Not used
Not used
Not used

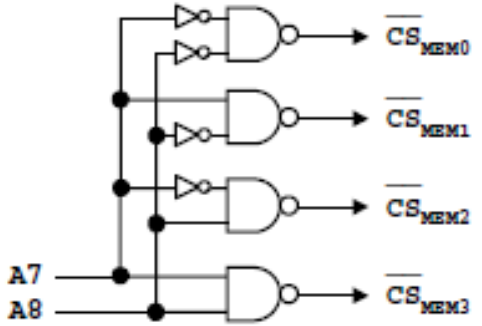
ARM

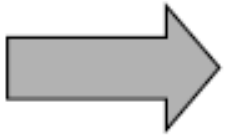
Partial Address Decoding

- Using the same requirements as before -

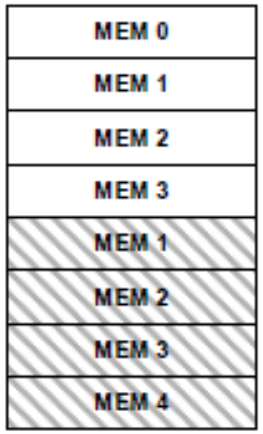
Not used | *Used for Address Decoding* | *Used to reference memory cells on each memory IC*

Device	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
MEM 0	X	0	0	X	X	X	X	X	X	X
MEM 1	X	0	1	X	X	X	X	X	X	X
MEM 2	X	1	0	X	X	X	X	X	X	X
MEM 3	X	1	1	X	X	X	X	X	X	X





Memory map



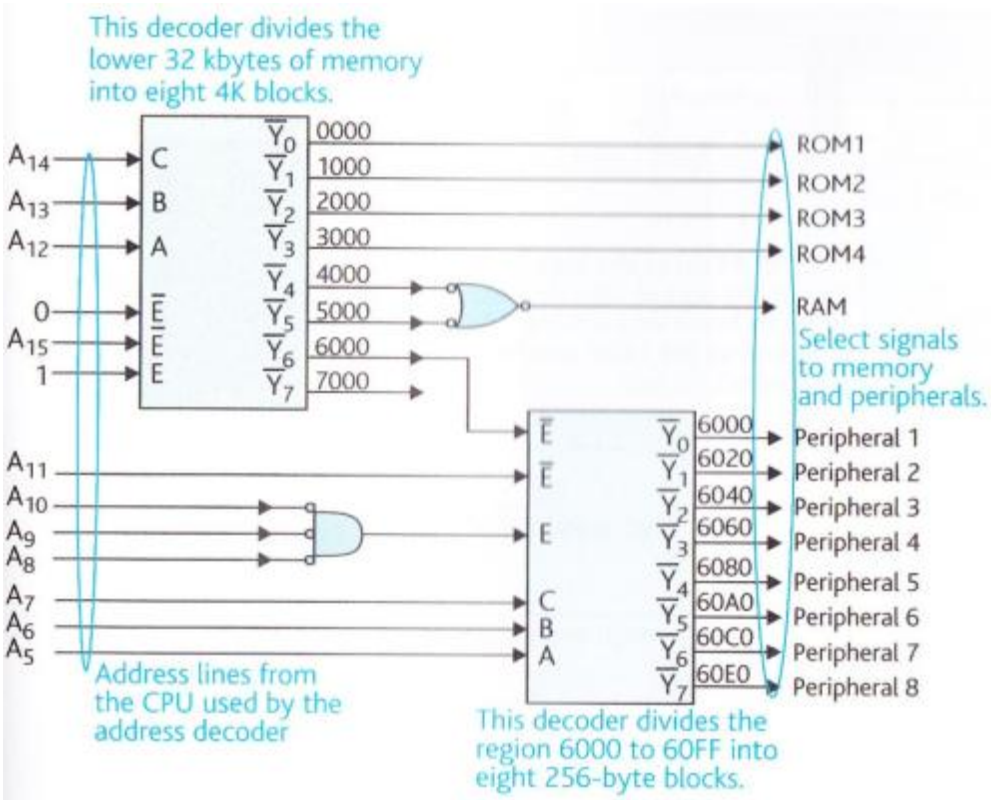
ARM

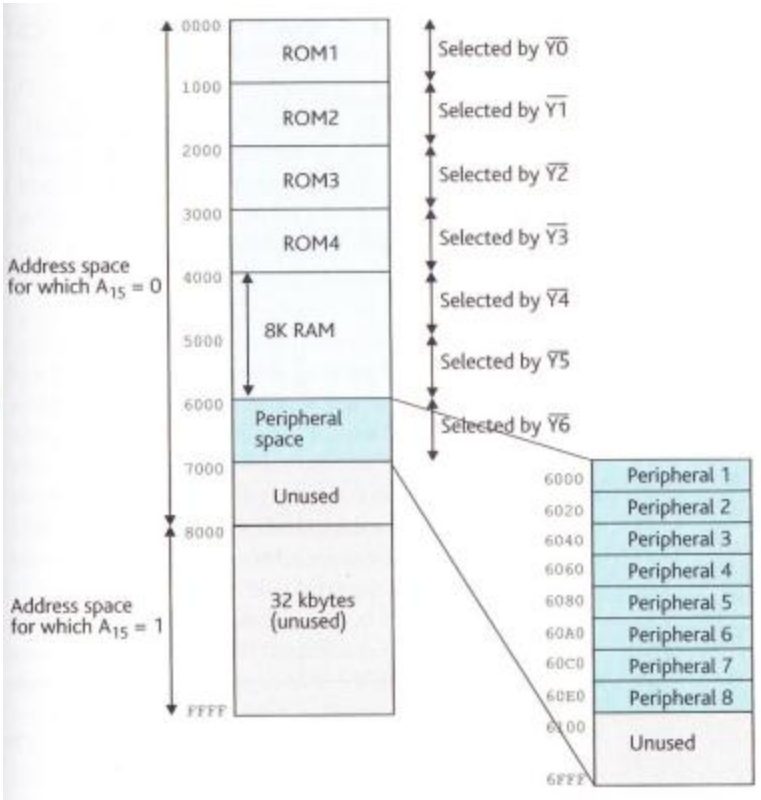
Implementing Address Decoders

- Discrete Logic
  - High Speed
  - High Chip Count
  - Lack Flexibility
- Data Decoders
  - Selection of devices is determined by the physical wiring
  - All memory blocks must have same size
- Programmable Read Only Memory (PROM)
  - Versatile, since the selection of the devices is determined by the programming
  - Memory blocks can be of different sizes
- Other Methods
  - FPGA
  - PLA

Using m-line to n-line Decoders

- Decoding using 3-to-8 decoders and discrete logic.





ARM

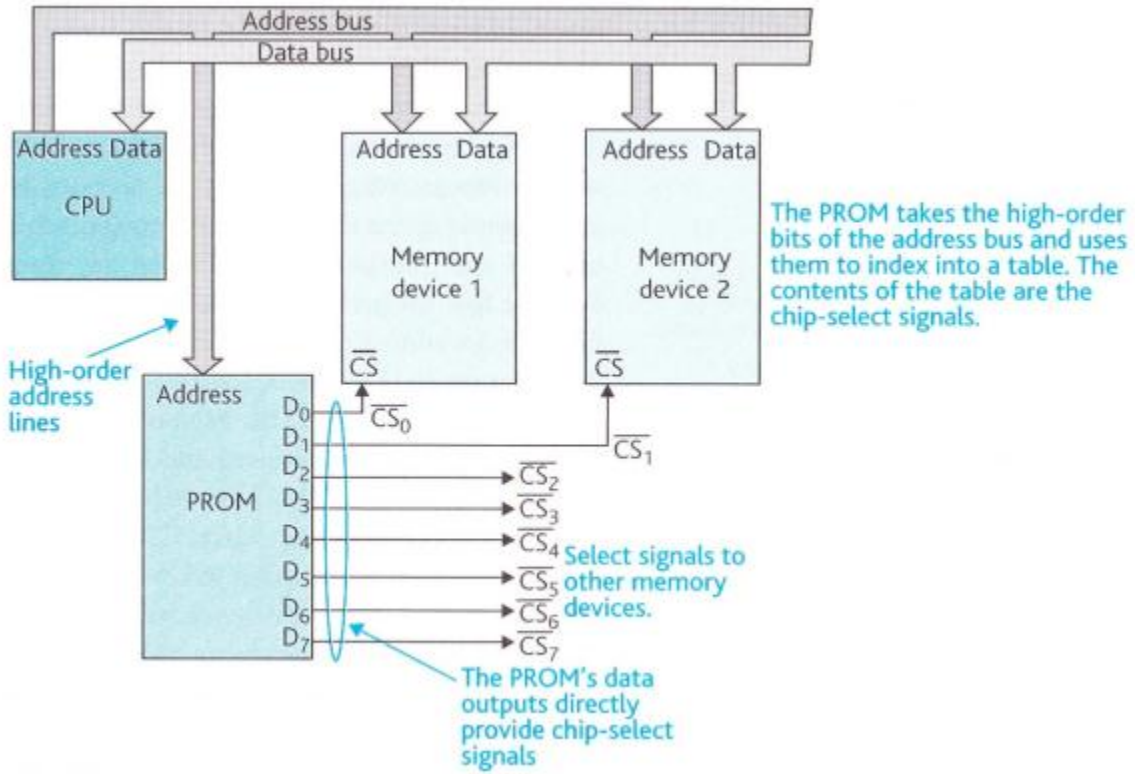
Using m-line to n-line Decoders

- Truth-Table for the earlier Decoding Design using Decoders

Device	Size	Address Range	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
ROM1	4K	0000–0FFF	0	0	0	0	x	x	x	x	x	x	x	x	x	x	x	x
ROM2	4K	1000–1FFF	0	0	0	1	x	x	x	x	x	x	x	x	x	x	x	x
ROM3	4K	2000–2FFF	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x
ROM4	4K	3000–3FFF	0	0	1	1	x	x	x	x	x	x	x	x	x	x	x	x
RAM	8K	4000–5FFF	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x	x
P1	32	6000–601F	0	1	1	0	0	0	0	0	0	0	0	x	x	x	x	x
P2	32	6020–603F	0	1	1	0	0	0	0	0	0	0	1	x	x	x	x	x
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
P8	32	60E0–60FF	0	1	1	0	0	0	0	0	1	1	1	x	x	x	x	x

Decoding using PROM

- Programmable Read-Only Memory chips hold look-up tables that would determine which $\overline{CS}$ to activate with any given address.
- The PROM's *n* address inputs select one of 2^n unique locations.



ARM

Decoding using PROM

- Truth-Table for the earlier Decoding Design using PROM

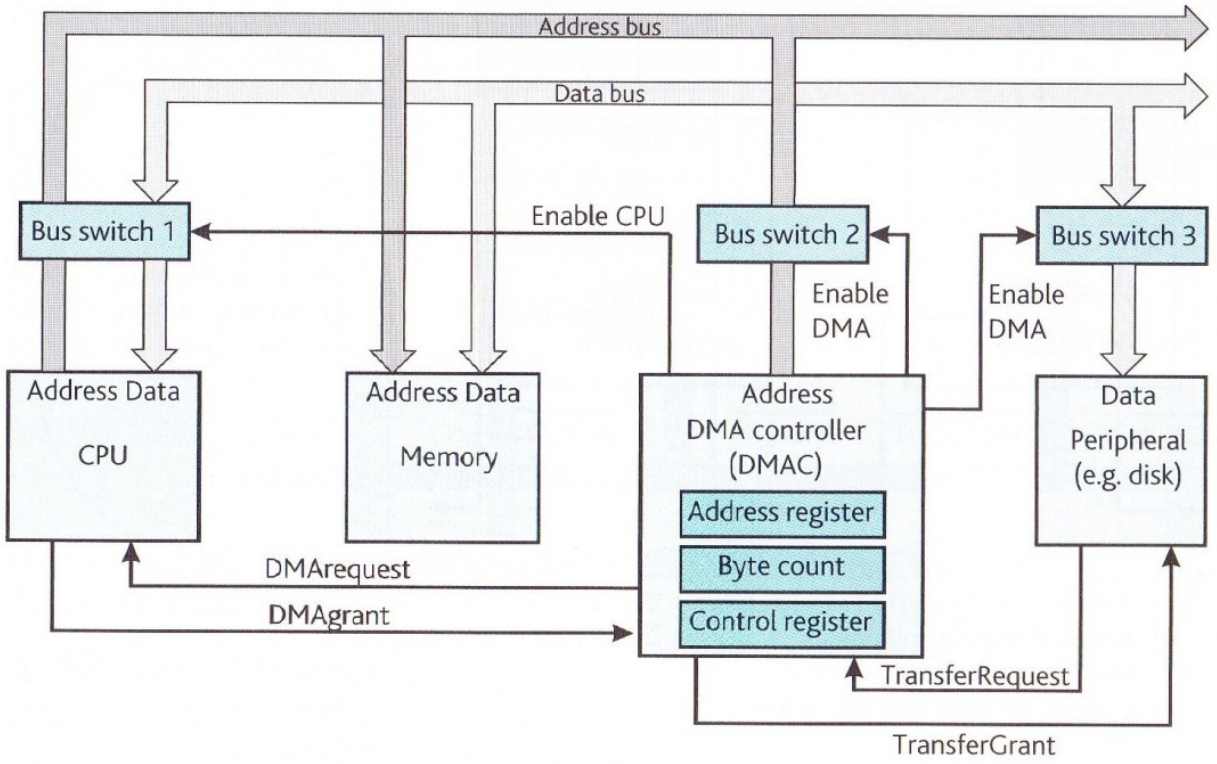
Inputs				Outputs								
A_{15}	A_{14}	A_{13}	A_{12}	CPU name	$\overline{CS0}$	$\overline{CS1}$	$\overline{CS2}$	$\overline{CS3}$	$\overline{CS4}$	$\overline{CS5}$	$\overline{CS6}$	$\overline{CS7}$
A_3	A_2	A_1	A_0	PROM name	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
				Range								
0	0	0	0	0000 to 0FFF	**0**	1	1	1	1	1	1	1
0	0	0	1	1000 to 1FFF	1	**0**	1	1	1	1	1	1
0	0	1	0	2000 to 2FFF	1	1	**0**	1	1	1	1	1
0	0	1	1	3000 to 3FFF	1	1	1	**0**	1	1	1	1
0	1	0	0	4000 to 4FFF	1	1	1	1	1	1	1	1
0	1	0	1	5000 to 5FFF	1	1	1	1	1	1	1	1
0	1	1	0	6000 to 6FFF	1	1	1	1	**0**	1	1	1
0	1	1	1	7000 to 7FFF	1	1	1	1	**0**	1	1	1
1	0	0	0	8000 to 8FFF	1	1	1	1	1	1	1	1
1	0	0	1	9000 to 9FFF	1	1	1	1	1	1	1	1
1	0	1	0	A000 to AFFF	1	1	1	1	1	1	1	1
1	0	1	1	B000 to BFFF	1	1	1	1	1	1	1	1
1	1	0	0	C000 to CFFF	1	1	1	1	1	**0**	1	1
1	1	0	1	D000 to DFFF	1	1	1	1	1	**0**	1	1
1	1	1	0	E000 to EFFF	1	1	1	1	1	**0**	1	1
1	1	1	1	F000 to FFFF	1	1	1	1	1	**0**	1	1

Direct Memory Access

- Direct Memory Access (DMA) moves data between a peripheral and the CPU's memory without the direct intervention of the CPU itself.
- Provides the fastest possible means of transferring data between an interface and memory.
- It requires no CPU overhead and leaves the CPU free to do useful work.
- Helps to increase data throughput.

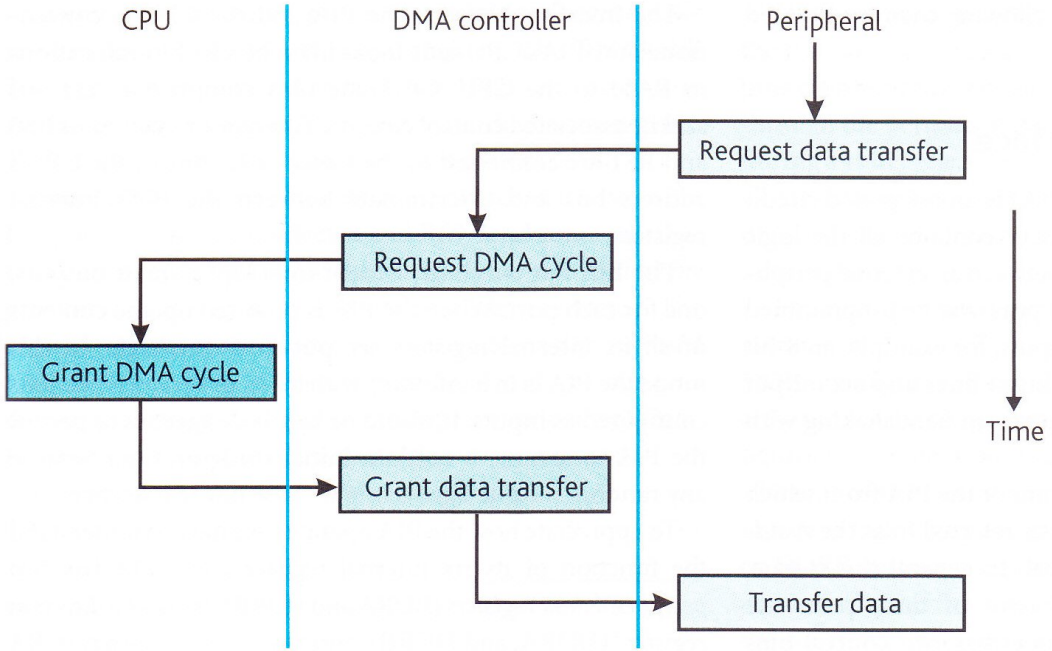
Direct Memory Access

- During normal operation of the computer, bus switch 1 is closed and bus switches 2 and 3 are open.
- The CPU control the buses, providing an address on the address bus and reading and writing data from memory via the data bus.



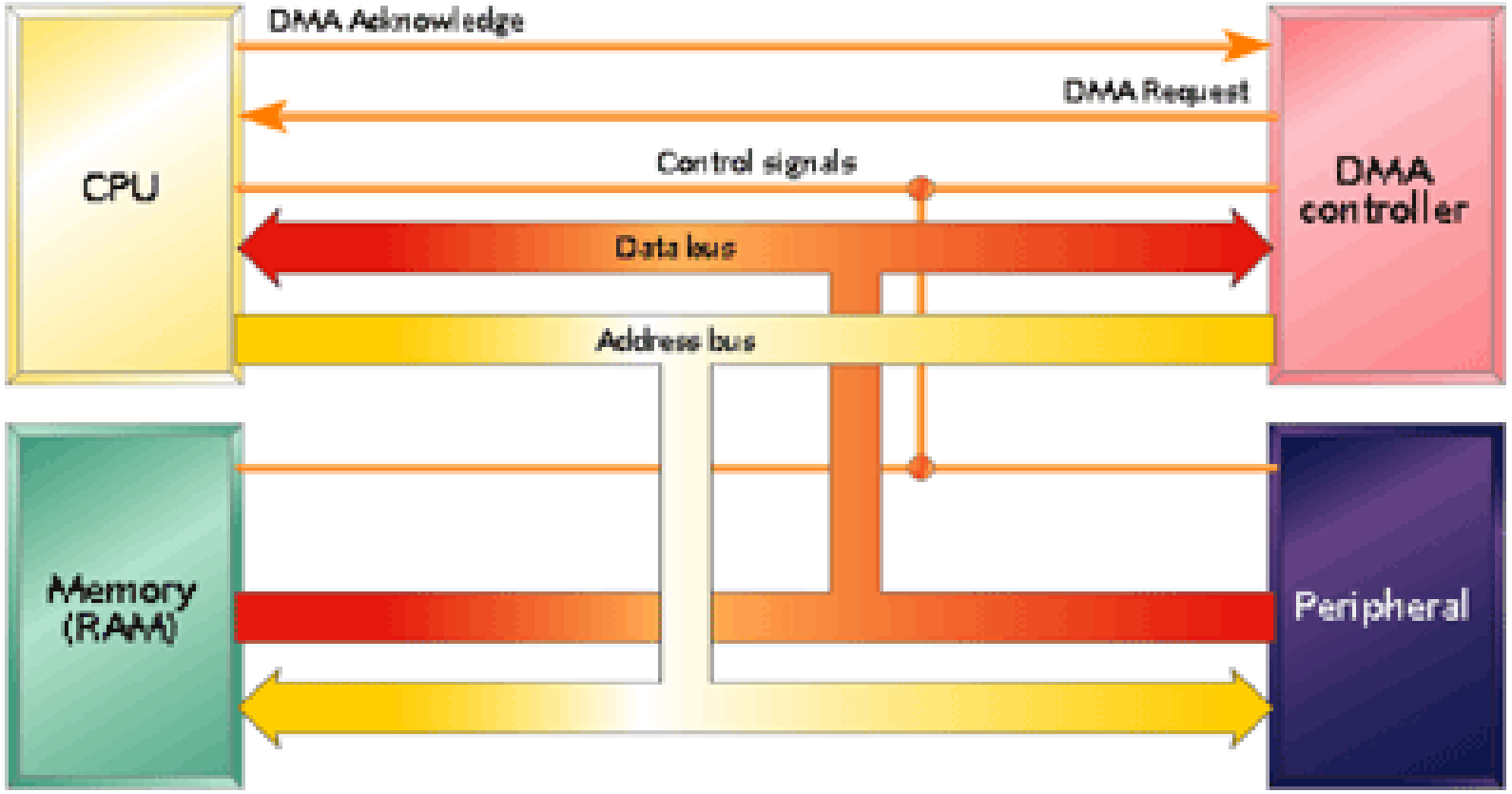
Direct Memory Access

- When a peripheral wishes to take part in an I/O transaction it asserts the TransferRequest input of the DMA controller (DMAC).
- In turn, the DMAC controller asserts the DMArequest (BusReq) to request control of the buses from the CPU.
- When the CPU returns DMAgrant (BusAck) to the DMAC, a DMA transfer takes place.
- Bus switch 1 is open and switches 2 and 3 are closed.
- The DMAC provides TransferGrant signal to the peripheral.
- The peripheral is able to communicate with the memory using the DMA
- When DMA operation is completed, DMAC hands control of the bus back to the CPU.



Direct Memory Access

- It is not necessary for the DMA to be initiated by the Peripheral.
- It can be initiated within the Processor itself.



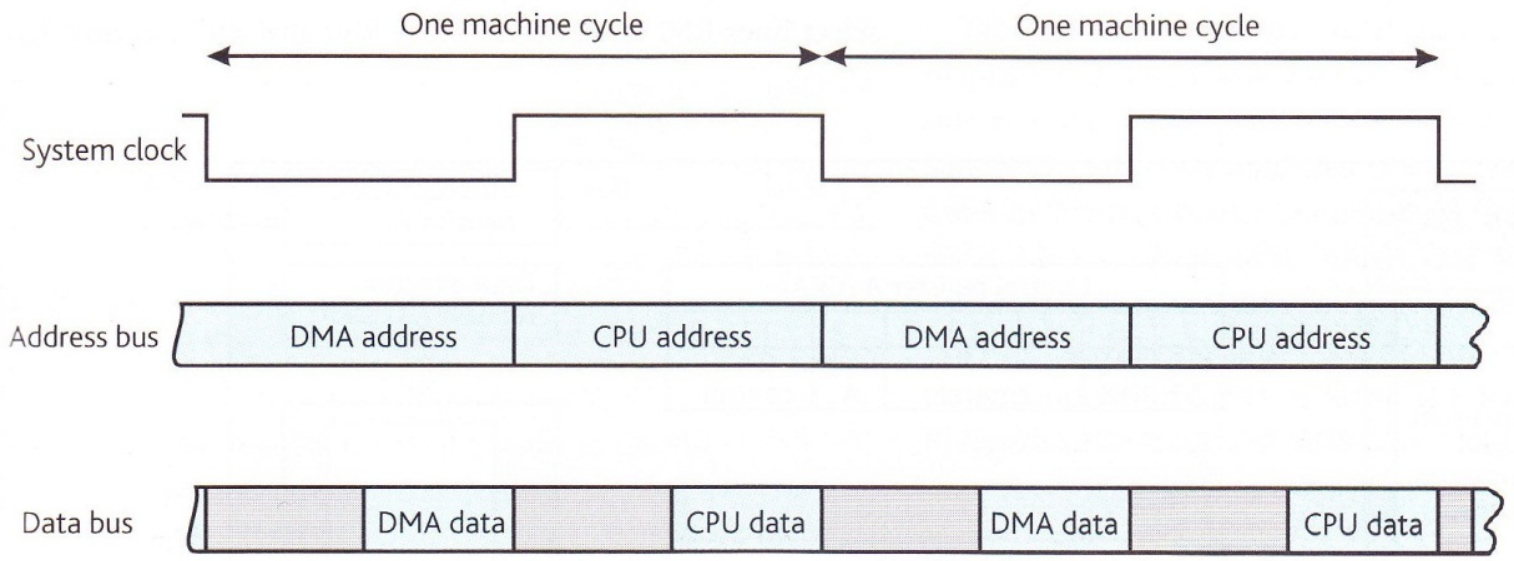
Direct Memory Access

- Many DMACs are able to handle several interfaces, which requires their registers to be duplicated.
- Each interface is referred to as a channel.
- DMA operates in one of two modes: burst mode or cycle stealing

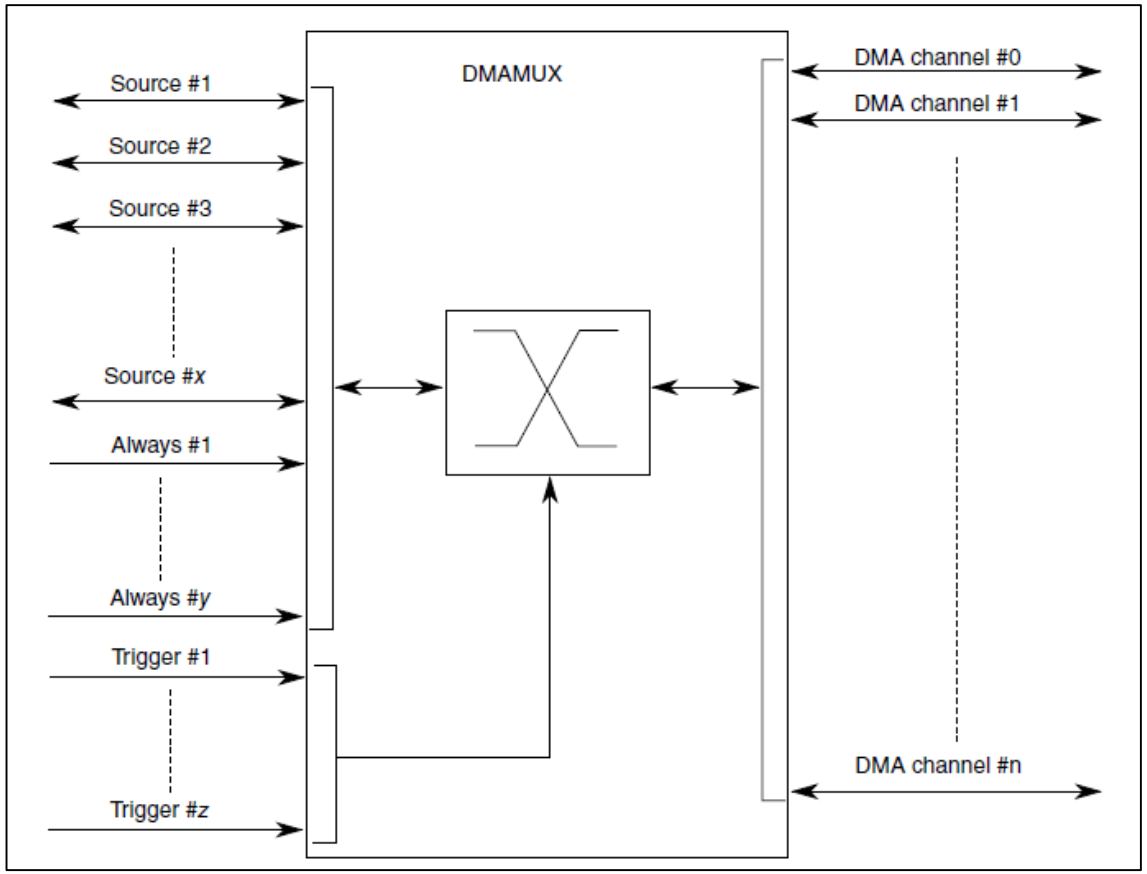
- Burst Mode
  DMA controller seizes the system bus for the duration of the data transfer operation.
  Allows data to be moved at the fastest possible rate within the memory/bus/interface timing requirements
  CPU is effectively halted in the burst mode because it cannot use its buses.

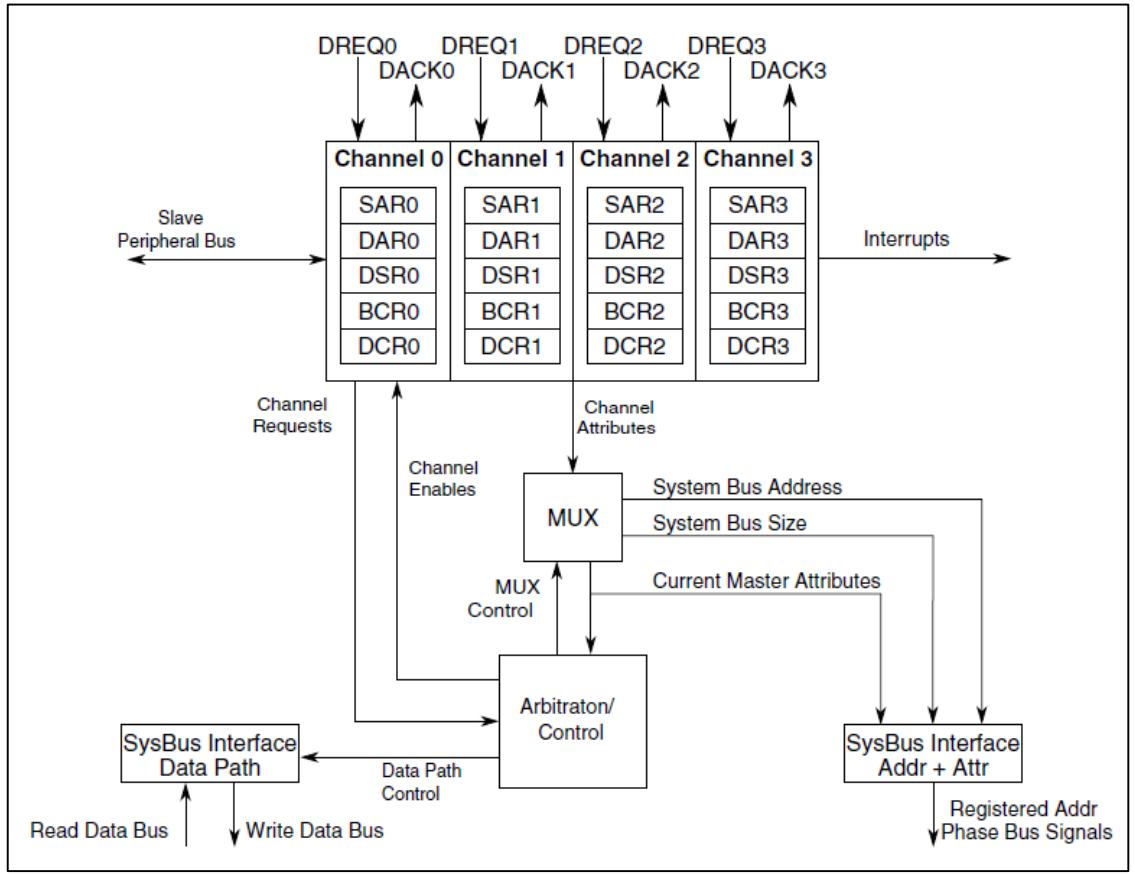
Direct Memory Access

- Cycle Steal Mode
  - DMA operations are interleaved with the processor's normal memory accesses.
  - DMA can take place when CPU is busy.
  - Transparent because transfer is invisible to the processor and no processing time is lost.



Let's Look at the DMA Block in the KL25Z





ARM

Watchdog Timers

- A system can fail due to many reasons
  - Priority Inversion
  - Deadlocks
  - Programming Error
  - Hardware Failure
  - Unexpected Events, etc.

- Even though strategies exist to avoid some of the problems, they may not necessarily be implemented due to expense, complexity or oversight.

- If continuous operation of the embedded application is critical and if the failure is only temporary, a '*watchdog timer*' can be used as a primitive recovery mechanism.

21

Watchdog Timers

- A watchdog timer is simply a hardware counter that counts down towards zero at a fixed rate.
- The software is expected to periodically restart the counter so that it never actually reaches zero.
- If the counter *does* reach zero, a failure is assumed to have occurred.
- The watchdog hardware then sends a reset signal to the CPU, causing the system to reinitialize.
- In safety-critical applications, the watchdog hardware must also be responsible for putting everything into a safe state, since the CPU itself may have failed.

22

Watchdog Timers

- In a real-time multithreaded application, it is important to that the watchdog timer verify more than just the fact that the CPU is executing instructions.
- The watchdog ISR could easily continue to work properly even in the presence of a deadlock or priority inversion, and thus would not verify that tasks are meeting their deadlines.
- Thus, strategies have to be implemented so that the ISR restarts the counter only after verifying that all task deadlines have been met.

23

Case Study

The Clementine

- In 1994, a deep space probe, the Clementine, was launched to make observations of the moon and a large asteroid (1620 Geographos).
- After months of operation, a software exception caused a control thruster to fire for 11 minutes, which depleted most of the remaining fuel and caused the probe to rotate at 80 RPM.
- Control was eventually regained, but it was too late to successfully complete the mission.

24

Case Study

Mars Pathfinder

- In July of 1997, a priority inversion occurred on the Mars Pathfinder mission, after the craft had landed on the Martian surface.
- A high priority communications task was forced to wait on a mutex held by a lower priority "science" task.
- The timing of the software was compromised, and a system reset issued by its watchdog timer brought the system back to normal operating conditions.
- On Earth, scientists were able to identify the problem and upload new code to fix the problem.
- Thus, the rest of the $265 million dollar mission could be completed successfully.

25

Case Study

Patriot Missile Bug

- During Operation Desert Shield, the US military deployed the Patriot Missile System.
- The system uses the velocity of its target and the current time to predict where the target will be from one instant to another.
- There was a bug in the targeting software, where over time, the internal clock would 'drift' further and further from the accurate time.
- The bug was known and simply fixed by regularly rebooting the system. However, the soldiers operating the system didn't know how to interpret 'regular' and left it on for 100 hours.
- An incoming missile was miscalculated to land in an open field due to the a drift of 0.34s.
- Eventually it hit a US airfield killing and injuring many soldiers.

26

Final Exam Matters

- LumiNUS Quiz
- All Chapters Covered.
  - HW Interfacing/Programming -> Conceptual
  - RTOS onwards -> Focus Area
- Short-Answer Type Questions
- Answer-Boxes with Word Limits are Intentional
- Analysis/Writing of Code

The End!

- We have completed the module! Hooray! ☺
- You have obtained extensive experience and knowledge in both Embedded Systems Development together with key RTOS concepts.
- If you are keen to be a TA for this module next semester, please fill-in the Google Form that will be sent out soon!
- We appreciate your feedback, which helps us improve
- We look forward to not just feedback for this module, but also feedback for the CEG programme.
- Thank You and Good Luck for the Exams! ☺

28