ARM CortexM3 – Programmers view and Development Environment

Hardware Software CoDesign

November 2011, December 2011

Agenda

ARM CortexM3 – Programmers view and Development Environment

1. Continued... Programmers view of CortexM3 (refer as M3)

2. Discussion on the Answering Machine Assignment (Group wise completion)

3. Mid Semester Paper distribution (left-overs and any doubts)

4. Introduction to Image Processing – Pixelization, Quantization

ARM CortexM3 – Programmers view

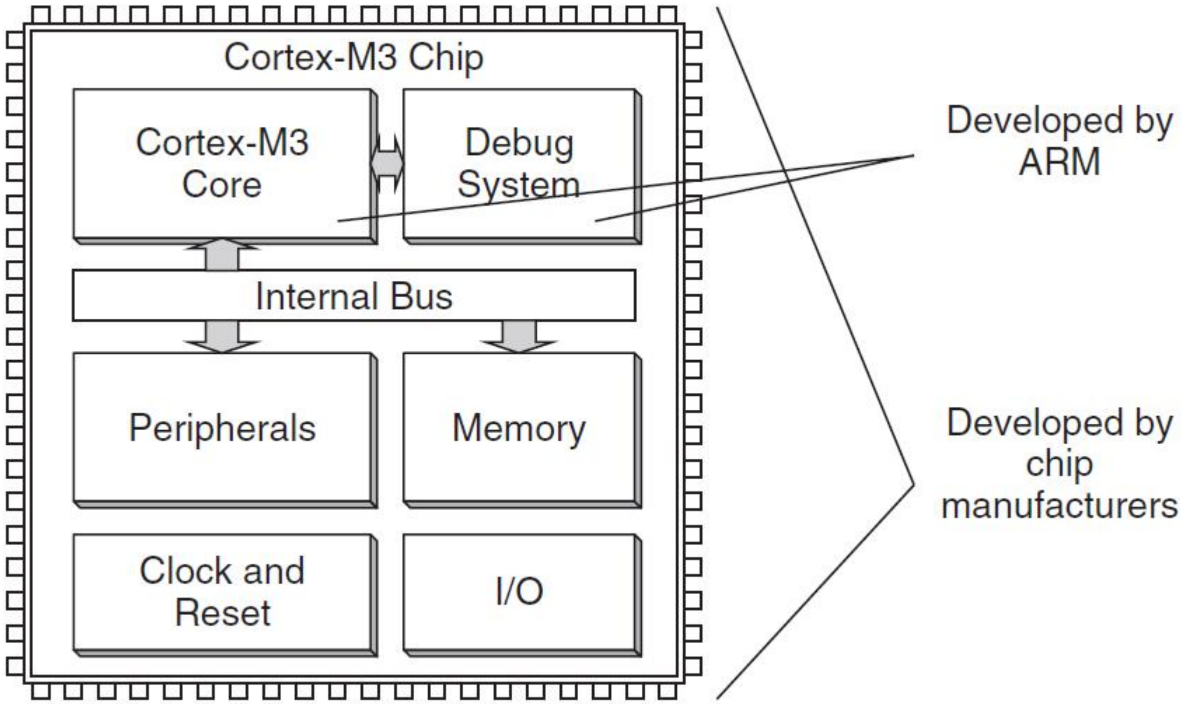
Introduction to ARM CortexM3

1. Primarily designed to target the 32bit Microcontoller market
2. Great performance at low cost and many new features available only in high-end processors
3. Enhanced determinism, guaranteeing that critical tasks and interrupts are serviced as quickly as possible, but in a "known" number of cycles.
4. Improve code density, ensuring that code fits even the smallest memory footprints
5. Ease of use, providing debugability and easy programmability for those applications which are migrating from 8, 16bit to 32bit.
6. Can be used in "device aggregation", where multiple traditional 8bit devices can get replaced by a single 32bit high performance device.
7. Through the compilers, the amount of code reuse across other ARM systems can take place.

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3

1. The use model for ARM and other integrators is :-



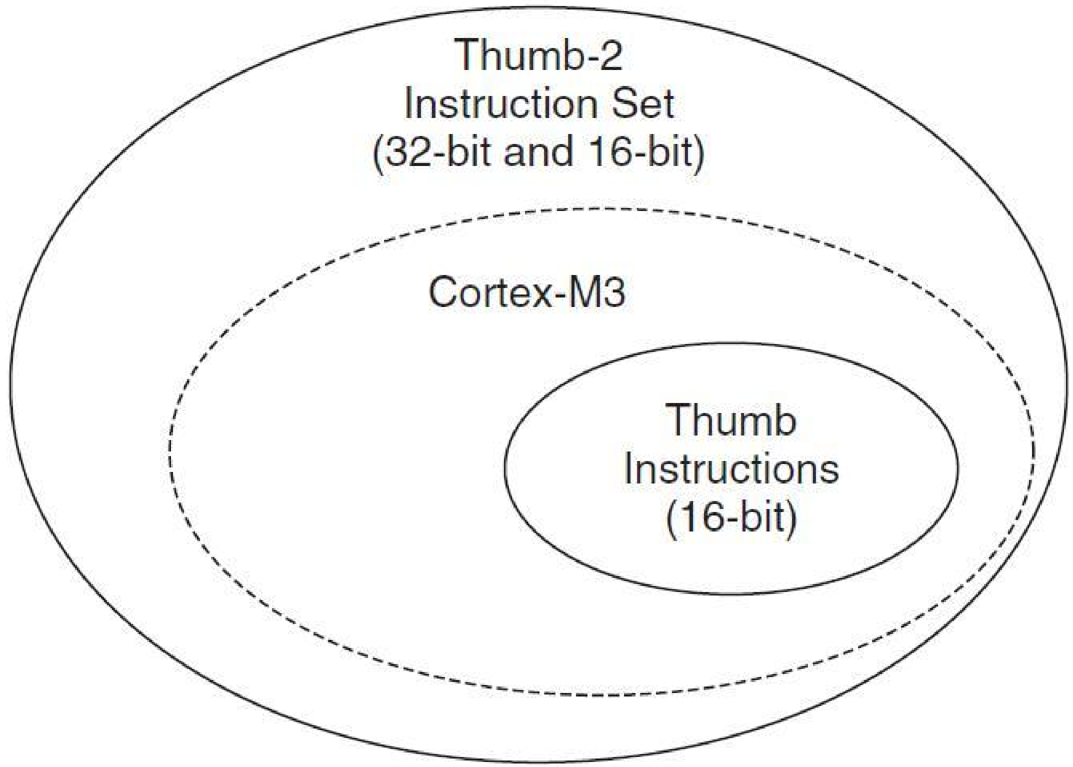
2. In general, ARM has come up with Cortex family like :-

- **A family** :: designed for high-performance *application* platforms
- **R family** :: designed for high-end embedded systems in which *Real-Time* performance is needed
- **M family** :: designed for deeply embedded *Microcontroller* systems

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – ARM and Thumb Instruction Sets

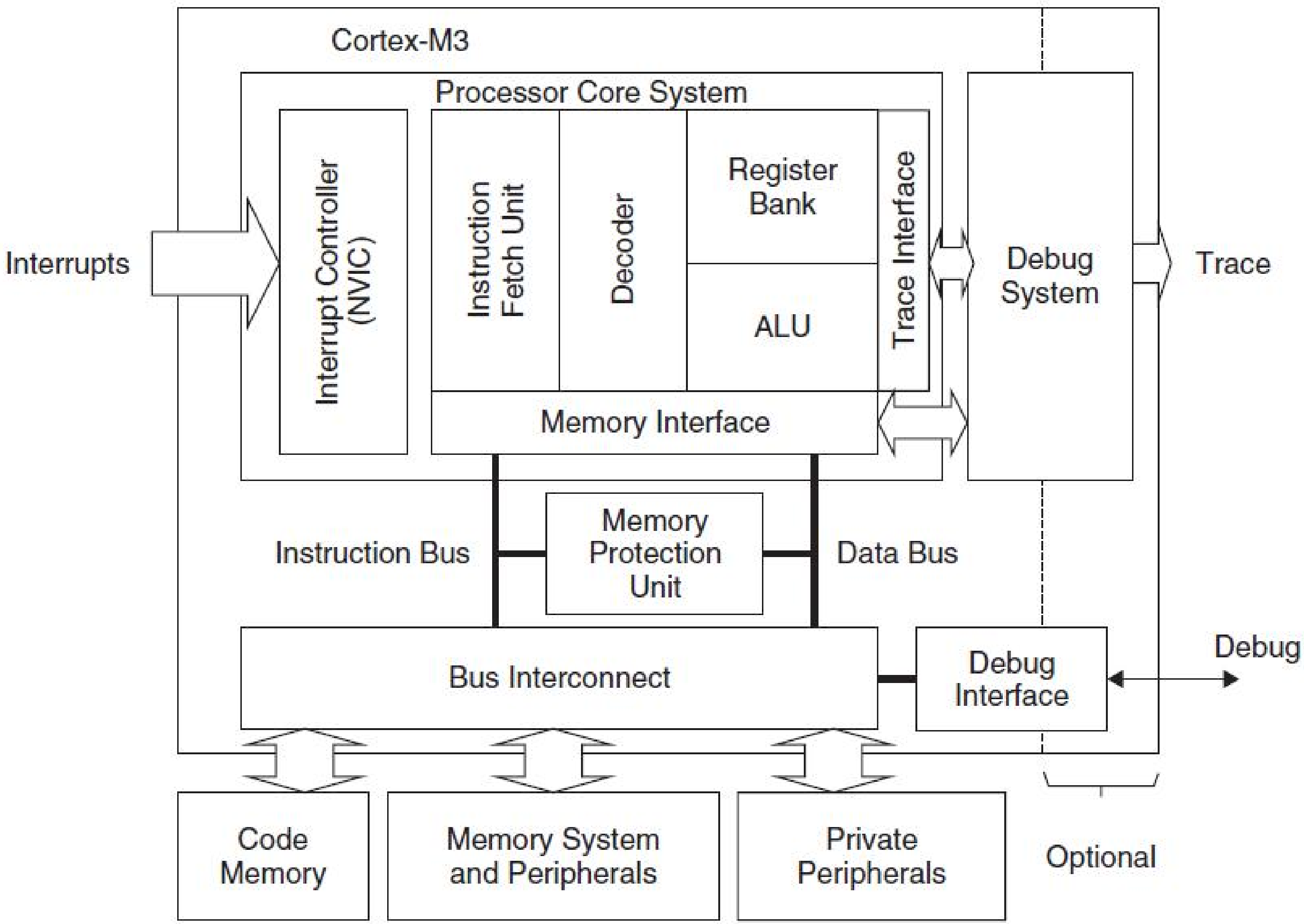
1. There are two different types of instruction sets
   (a) 32bit called *ARM* instruction set
   (b) 16bit called *Thumb* instruction set
2. During program execution, the processor can be dynamically switched between the ARM state or Thumb state to use either of the instruction sets
3. The Thumb instruction set provides only a subset of the ARM instructions, but can provide high code density.
4. M3 processor supports only the Thumb-2 (and traditional Thumb) instruction set. It uses Thumb-2 instruction set for all operations



The Relationship Between the Thumb-2 Instruction Set and the Thumb Instruction Set

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – Basic Block Diagram



A Simplified View of the Cortex-M3

1. The processor has a Harvard architecture, ie., has a separate instruction bus and data bus. However, the instruction and data buses share the same memory space (unified memory system). i.e, Cannot get 8GB space just because there are separate bus interfaces.

2. *GROUP DISCUSSION* :: How does Harvard architecture help M3 ?

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – General Purpose Register Set

1. The M3 has GP registers *R0 to R15*
2. *General Purpose Registers* :: R0 to R12
3. *Stack Pointer* :: R13 $\rightarrow$ banked R13 register
   (a) *Main Stack Pointer MSP* - Default StackPointer, used by the OS kernel and exception handlers
   (b) *Process Stack Pointer PSP* - Used by the user application code
4. *Link Register* :: R14 $\rightarrow$ When a subroutine is called, the return address is stored in the link register.
5. *Program Counter* :: R15 $\rightarrow$ The current program address. This register can be written to control the program flow

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – Operation Modes

1. The M3 has 2 Operation modes and 2 privilege levels

2. *Operation Modes* :: What kind of operation the M3 is doing. Whether the processor is running a normal program or running an exception handler like an interrupt handler or system exception handler.

   (a) *Thread Mode* :: Thread mode is entered on reset, and can be entered as a result of an exception return. Privileged and User code can run in Thread mode.

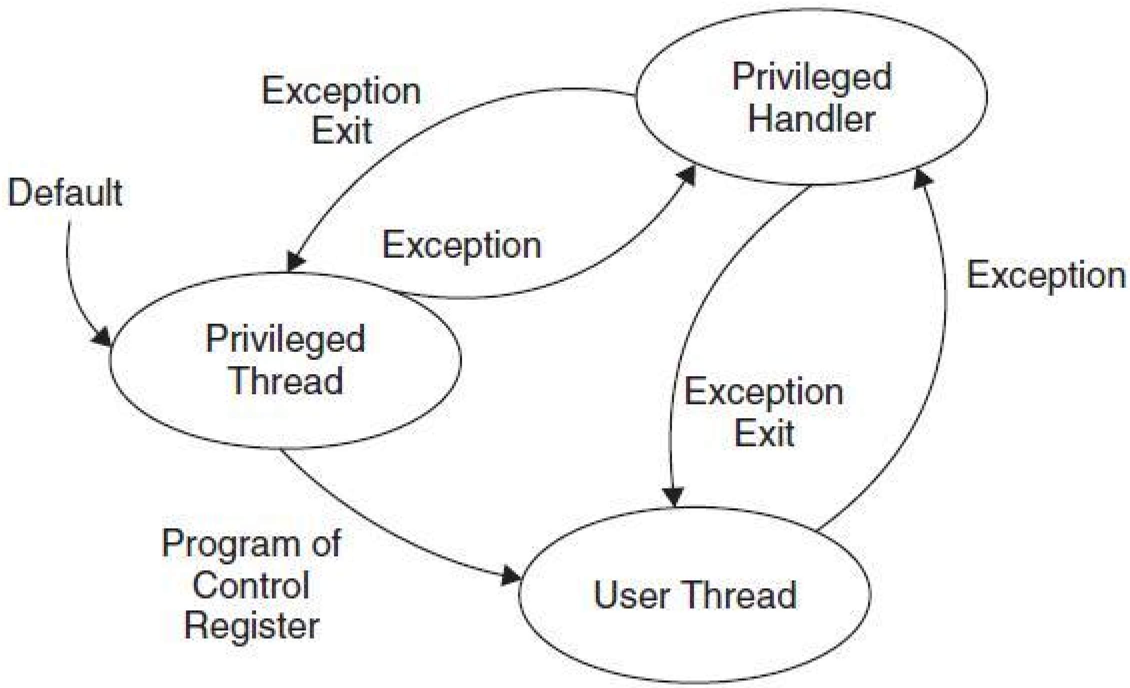
   (b) *Handler Mode* :: Handler mode is entered as a result of an exception. All code is privileged in Handler mode.

3. *Privilege Levels* :: provide a mechanizm for safeguarding memory acces to critical regions as well as provide a basic security model.

   (a) *Privilege Level* ::

   (b) *User Level* ::

	Privileged	***User***
When running an exception	Handle Mode	
When running main program	Thread Mode	Thread Mode



Allowed Operation Mode Transitions

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – BuiltIn Nested Vectored Interrupt Controller

1. The M3 Processor includes an Interrupt Controller called the *NVIC (Nested Vectored Interrupt Controller* with following main features :-

   (a) *Nested Interrupt Support* :: All interrupts and most of the system exceptions can be programmed to different priority levels. When an interrupt occurs, the NVIC compares the priority of this interrupt to the current running priority level. If the priority of the new interrupt is higher than the current level, the interrupt handler of the new interrupt will override the current running task.

   (b) *Vectored Interrupt Support* :: When an interrupt is accepted, the starting address of the ISR is located from a vector table in memory. There is no need to use software to determine and branch to the starting address of the ISR. Thus it takes less time to process the interrupt request.

   (c) *Dynamic Priority changes Support* :: Priority levels of interrupts can be changed by software during run time. Interrupts that are being serviced are blocked from further activation until the ISR is completed, so their priority can be changed without risk of accidental re-entry.

   (d) *Reduction of Interrupt Latency* :: M3 includes automatic saving and restoring some register contents, reducing delay in switching from one ISR to another and handling late arrival interrupts.

   (e) *Interrupt Masking* :: Interrupts and system exceptions can be masked based on their priority level or masked completly using interrupt masking registers BASEPRI, PRIMASK, FAULTMASK. They can be used to ensure that time-critical tasks can be finished on time without being interrupted.

Cortex-M3 Exception Types

Exception Number	Exception Type	Priority (Default to 0 if Programmable)	Description
0	NA	NA	No exception running
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus fault	Programmable	Bus error (Prefetch Abort or Data Abort)
6	Usage fault	Programmable	Exceptions due to program error
7–10	Reserved	NA	Reserved
11	SVCall	Programmable	System service call
12	Debug monitor	Programmable	Debug monitor (break points, watchpoints, or external debug request)
13	Reserved	NA	Reserved
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System tick timer
16	IRQ #0	Programmable	External interrupt #0
17	IRQ #1	Programmable	External interrupt #1
...	...	...	...
255	IRQ #239	Programmable	External interrupt #239

ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – Memory Map

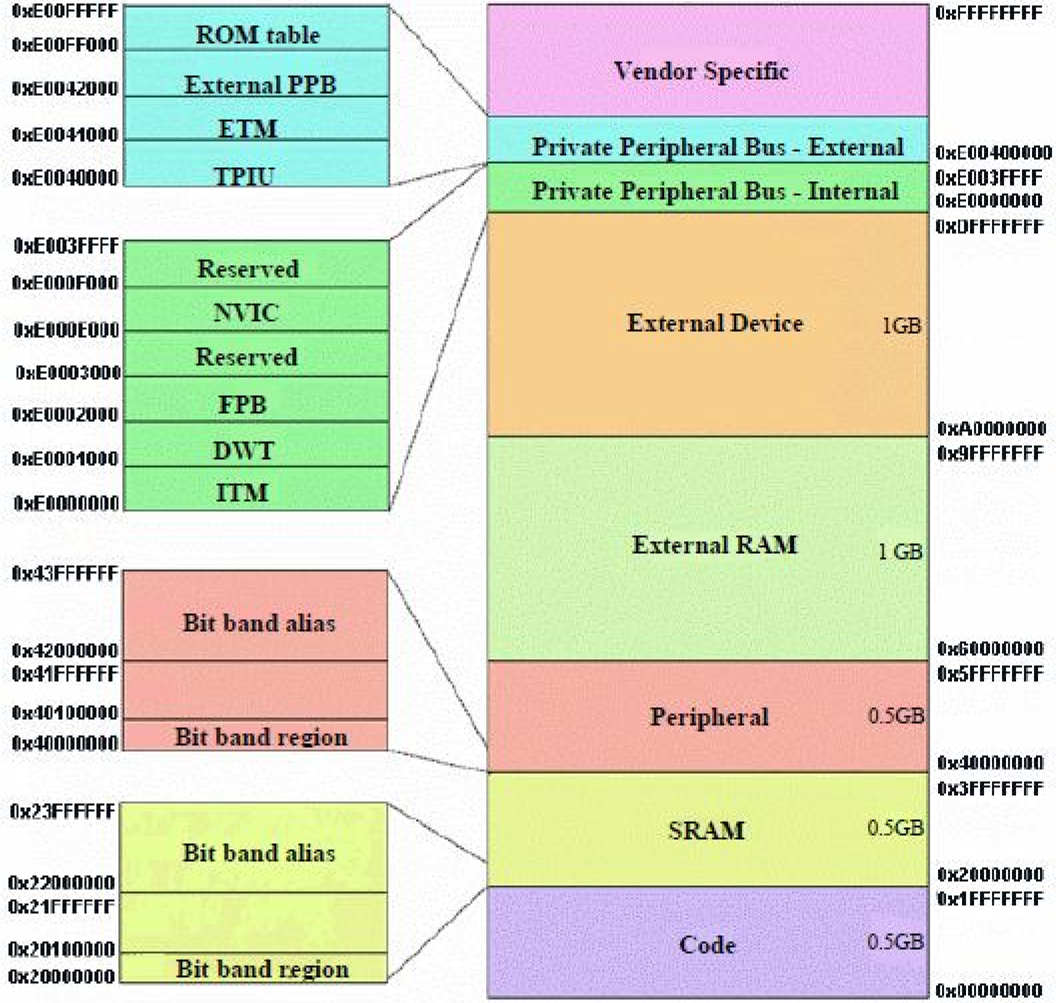
1. M3 has a predefined memory map. This allows the built-in peripherals, such as NVIC, and debug components to be accessed by simple memory access instructions.

2. This allows most system features through normal C program code.

3. Allows optimization for ease of integration and re-use

4. M3 has an optional Memory Protection Unit (MPU). The MPU is setup by an OS, allowing data used by privileged code to be protected from User programs. The MPU can be used to make memory regions ReadOnly to prevent accidental erasing of data, or to isolate memory regions between different tasks in a multi-tasking system. Overall, helps in making systems more robost and reliable.

Figure 4. The memory map

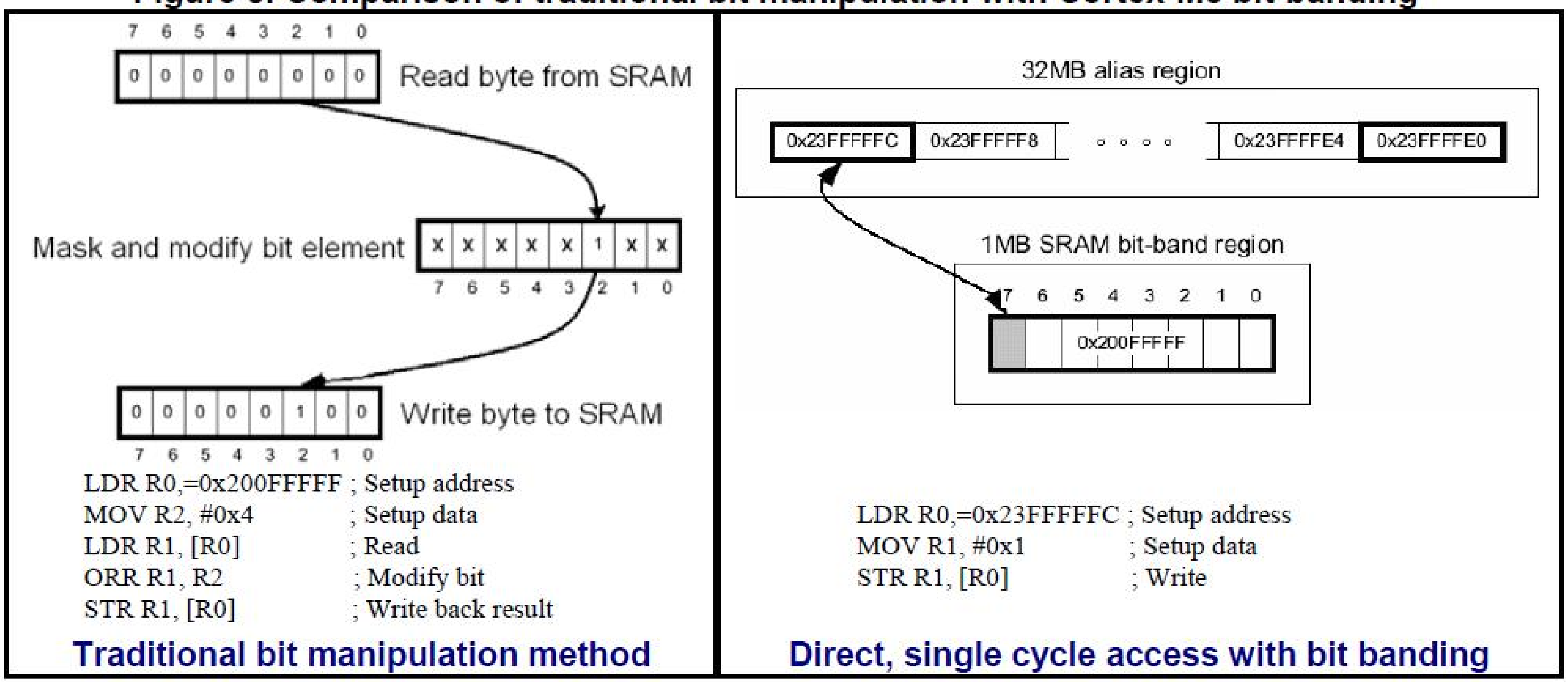


ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – Memory Map – Bit Banding

The Cortex-M3 processor enables direct access to single bits of data in simple systems by implementing a technique called bit-banding (Figure 5). The memory map includes two 1MB bit-band regions in the SRAM and peripheral space that map on to 32MB of alias regions. Load/store operations on an address in the alias region directly get translated to an operation on the bit aliased by that address. Writing to an address in the alias region with the least-significant bit set writes a 1 to the bit-band bit and writing with the least-significant bit cleared writes a 0 to the bit. Reading the aliased address directly returns the value in the appropriate bit-band bit. Additionally, this operation is atomic and cannot be interrupted by other bus activities.

Figure 5. Comparison of traditional bit manipulation with Cortex-M3 bit-banding



ARM CortexM3 – Programmers view

Introduction to ARM CortexM3 – Instruction Set

1. Basic syntax used is :: `opcode operand1, operand2, ...  ; Comments`
2. Bringup the TRM of M3 or GuideToArmCortexM3 (pg 71). Following categories :-
3. 16-bit Data processing instructions
4. 16-bit Branch instructions
5. 16-bit Load and Store instructions
6. Other 16-bit instructions
7. Same as above with 32-bit instructions

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q	Application PSR										
IPSR	Interrupt PSR											Exception Number				
EPSR	Execution PSR					ICI/IT	T			ICI/IT						

Program Status Registers (PSRs) in the Cortex-M3

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number				

Combined Program Status Registers (xPSR) in the Cortex-M3

Bit	**Description**
N	Negative
Z	Zero
C	Carry/borrow
V	Overflow
Q	Sticky saturation flag
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit
T	Thumb state, always 1; trying to clear this bit will cause a fault exception
Exception Number	Indicates which exception the processor is handling

Bit Fields in Cortex-M3 Program Status Registers

- Z (Zero) flag: This flag is set when the result of an instruction has a zero value or when a comparison of two data returns an equal result.
- N (Negative) flag: This flag is set when the result of an instruction has a negative value (bit 31 is 1).
- C (Carry) flag: This flag is for unsigned data processing—for example, in add (ADD) it is set when an overflow occurs; in subtract (SUB) it is set when a borrow did not occur (borrow is the invert of carry).
- V (Overflow) flag: This flag is for signed data processing; for example, in an add (ADD), when two positive values added together produce a negative value, or when two negative values added together produce a positive value.

16-Bit Data Processing Instructions

Instruction	Function
ADC	Add with carry
ADD	Add
AND	Logical AND
ASR	Arithmetic shift right
BIC	Bit clear (Logical AND one value with the logic inversion of another value)
CMN	Compare negative (compare one data with two's complement of another data and update flags)
CMP	Compare (compare two data and update flags)
CPY	Copy (available from architecture v6; move a value from one high or low register to another high or low register)
EOR	Exclusive OR
LSL	Logical shift left
LSR	Logical shift right
MOV	Move (can be used for register-to-register transfers or loading immediate data)
MUL	Multiply
MVN	Move NOT (obtain logical inverted value)
NEG	Negate (obtain two's complement value)
ORR	Logical OR
ROR	Rotate right
SBC	Subtract with carry
SUB	Subtract
TST	Test (use as logical AND; Z flag is updated but AND result is not stored)
REV	Reverse the byte order in a 32-bit register (available from architecture v6)
REVH	Reverse the byte order in each 16-bit half word of a 32-bit register (available from architecture v6)
REVSH	Reverse the byte order in the lower 16-bit half word of a 32-bit register and sign extends the result to 32 bits. (available from architecture v6)
SXTB	Signed extend byte (available from architecture v6)
SXTH	Signed extend half word (available from architecture v6)

16-Bit Branch Instructions

Instruction	Function
B	Branch
B<cond>	Conditional branch
BL	Branch with link; call a subroutine and store the return address in LR
BLX	Branch with link and change state (BLX <reg> only)[1]
CBZ	Compare and branch if zero (architecture v7)
CBNZ	Compare and branch if nonzero (architecture v7)
IT	IF-THEN (architecture v7)

16-Bit Load and Store Instructions

Instruction	Function
LDR	Load word from memory to register
LDRH	Load half word from memory to register
LDRB	Load byte from memory to register
LDRSH	Load half word from memory, sign extend it, and put it in register
LDRSB	Load byte from memory, sign extend it, and put it in register
STR	Store word from register to memory
STRH	Store half word from register to memory
STRB	Store byte from register to memory
LDMIA	Load multiple increment after
STMIA	Store multiple increment after
PUSH	Push multiple registers
POP	Pop multiple registers

Other 16-Bit Instructions

Instruction	Function
SVC	System service call
BKPT	Breakpoint; if debug is enabled, will enter debug mode (halted), or if debug monitor exception is enabled, will invoke the debug exception; otherwise it will invoke a fault exception
NOP	No operation
CPSIE	Enable PRIMASK (CPSIE i)/FAULTMASK (CPSIE f) register (set the register to 0)
CPSID	Disable PRIMASK (CPSID i)/ FAULTMASK (CPSID f) register (set the register to 1)

32-Bit Data Processing Instructions

Instruction	Function
ADC	Add with carry
ADD	Add
ADDW	Add wide (#immed_12)
AND	Logical AND
ASR	Arithmetic shift right
BIC	Bit clear (logical AND one value with the logic inversion of another value)
BFC	Bit field clear
BFI	Bit field insert

(Continued)

Instruction	Function
CMN	Compare negative (compare one data with two's complement of another data and update flags)
CMP	Compare (compare two data and update flags)
CLZ	Count lead zero
EOR	Exclusive OR
LSL	Logical shift left
LSR	Logical shift right
MLA	Multiply accumulate
MLS	Multiply and subtract
MOV	Move
MOVW	Move wide (write a 16-bit immediate value to register)
MOVT	Move top (write an immediate value to the top half word of destination reg)
MVN	Move negative
MUL	Multiply
ORR	Logical OR
ORN	Logical OR NOT
RBIT	Reverse bit
REV	Byte reserve word
REVH/REV16	Byte reverse packed half word
REVSH	Byte reverse signed half word
ROR	Rotate right register
RSB	Reverse subtract
RRX	Rotate right extended
SBFX	Signed bit field extract
SDIV	Signed divide
SMLAL	Signed multiply accumulate long
SMULL	Signed multiply long
SSAT	Signed saturate
SBC	Subtract with carry
SUB	Subtract
SUBW	Subtract wide (#immed_12)
SXTB	Sign extend byte
TEQ	Test equivalent (use as logical exclusive OR; flags are updated but result is not

Instruction	Function
TST	Test (use as logical AND; Z flag is updated but AND result is not stored)
UBFX	Unsigned bit field extract
UDIV	Unsigned divide
UMLAL	Unsigned multiply accumulate long
UMULL	Unsigned multiply long
USAT	Unsigned saturate
UXTB	Unsigned extend byte
UXTH	Unsigned extend half word

32-Bit Load and Store Instructions

Instruction	Function
LDR	Load word data from memory to register
LDRB	Load byte data from memory to register
LDRH	Load half word data from memory to register
LDRSB	Load byte data from memory, sign extend it, and put it to register
LDRSH	Load half word data from memory, sign extend it, and put it to register
LDM	Load multiple data from memory to registers
LDRD	Load double word data from memory to registers
STR	Store word to memory
STRB	Store byte data to memory
STRH	Store half word data to memory
STM	Store multiple words from registers to memory
STRD	Store double word data from registers to memory
PUSH	Push multiple registers
POP	Pop multiple registers

32-Bit Branch Instructions

Instruction	Function
B	Branch
BL	Branch and link
TBB	Table branch byte; forward branch using a table of single byte offset
TBH	Table branch half word; forward branch using a table of half word offset

Other 32-Bit Instructions

Instruction	Function
LDREX	Exclusive load word
LDREXH	Exclusive load half word
LDREXB	Exclusive load byte
STREX	Exclusive store word
STREXH	Exclusive store half word
STREXB	Exclusive store byte
CLREX	Clear the local exclusive access record of local processor
MRS	Move special register to general-purpose register
MSR	Move to special register from general-purpose register
NOP	No operation
SEV	Send event
WFE	Sleep and wake for event
WFI	Sleep and wake for interrupt
ISB	Instruction synchronization barrier
DSB	Data synchronization barrier
DMB	Data memory barrier

Examples of Arithmetic Instructions

Instruction	Operation

`ADD Rd, Rn, Rm ; Rd = Rn + Rm`
`ADD Rd, Rm ; Rd = Rd + Rm`
`ADD Rd, #immed ; Rd = Rd + #immed`
	ADD operation

`ADC Rd, Rn, Rm ; Rd = Rn + Rm + carry`
`ADC Rd, Rm ; Rd = Rd + Rm + carry`
`ADC Rd, #immed ; Rd = Rd + #immed +`
`; carry`
	ADD with carry

`ADDW Rd, Rn,#immed ; Rd = Rn + #immed`	ADD register with 12-bit immediate value

`SUB Rd, Rn, Rm ; Rd = Rn - Rm`
`SUB Rd, #immed ; Rd = Rd - #immed`
`SUB Rd, Rn,#immed ; Rd = Rn -#immed`
	SUBTRACT

`SBC Rd, Rm ; Rd = Rd - Rm -`
`; carry flag`
`SBC.W Rd, Rn, #immed ; Rd = Rn - #immed -`
`; carry flag`
`SBC.W Rd, Rn, Rm ; Rd = Rn - Rm -`
`; carry flag`
	SUBTRACT with borrow (carry)

`RSB.W Rd, Rn, #immed ; Rd = #immed -Rn`
`RSB.W Rd, Rn, Rm ; Rd = Rm - Rn`
`MUL Rd, Rm ; Rd = Rd * Rm`
`MUL.W Rd, Rn, Rm ; Rd = Rn * Rm`
	Reverse subtract
	
	Multiply

`UDIV Rd, Rn, Rm ; Rd = Rn /Rm` `SDIV Rd, Rn, Rm ; Rd = Rn /Rm`	Unsigned and signed divide

Logic Operation Instructions

<table>
<tr><th>Instruction</th><th>Operation</th></tr>
<tr><td><code>AND   Rd, Rn   ; Rd = Rd & Rn</code>
<code>AND.W Rd, Rn,#immed; Rd = Rn & #immed</code>
<code>AND.W Rd, Rn, Rm   ; Rd = Rn & Rd</code></td><td>Bitwise AND</td></tr>
<tr><td><code>ORR   Rd, Rn   ; Rd = Rd | Rn</code>
<code>ORR.W Rd, Rn,#immed; Rd = Rn | #immed</code>
<code>ORR.W Rd, Rn, Rm   ; Rd = Rn | Rd</code></td><td>Bitwise OR</td></tr>
<tr><td><code>BIC   Rd, Rn   ; Rd = Rd & (~Rn)</code>
<code>BIC.W Rd, Rn,#immed; Rd = Rn &(~#immed)</code>
<code>BIC.W Rd, Rn, Rm   ; Rd = Rn &(~Rd)</code></td><td>Bit clear</td></tr>
<tr><td><code>ORN.W Rd, Rn,#immed; Rd = Rn |(~#immed)</code>
<code>ORN.W Rd, Rn, Rm   ; Rd = Rn |(~Rd)</code></td><td>Bitwise OR NOT</td></tr>
<tr><td><code>EOR   Rd, Rn   ; Rd = Rd ^ Rn</code>
<code>EOR.W Rd, Rn,#immed; Rd = Rn | #immed</code>
<code>EOR.W Rd, Rn, Rm   ; Rd = Rn | Rd</code></td><td>Bitwise Exclusive OR</td></tr>
</table>

Shift and Rotate Instructions

Instruction	Operation

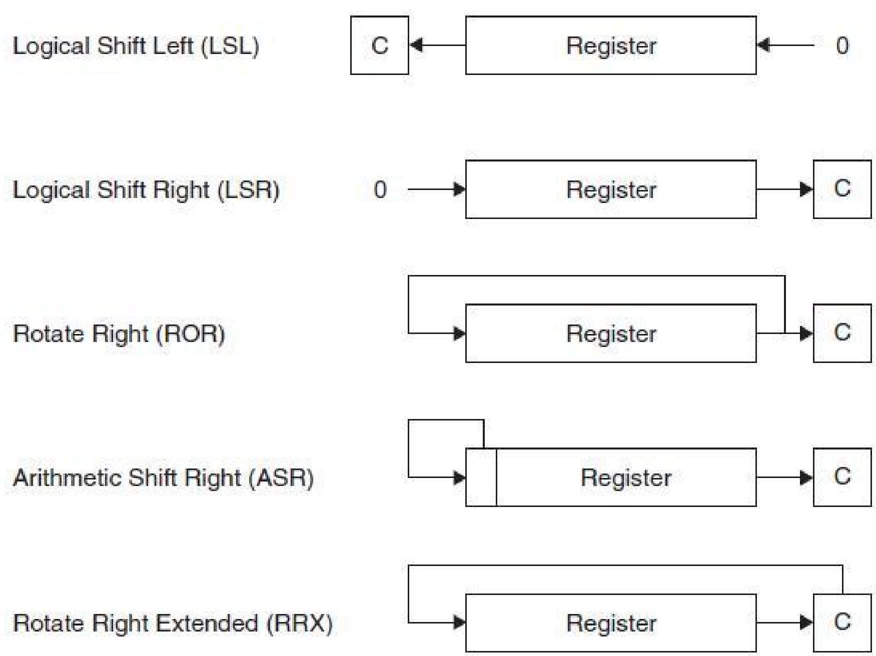
`ASR Rd, Rn,#immed; Rd = Rn >> immed`
`ASR Rd, Rn ; Rd = Rd >> Rn`
`ASR.W Rd, Rn, Rm ; Rd = Rn >> Rm`
	Arithmetic shift right

Instruction	Operation

`LSL Rd, Rn,#immed; Rd = Rn << immed`
`LSL Rd, Rn ; Rd = Rd << Rn`
`LSL.W Rd, Rn, Rm ; Rd = Rn << Rm`
	Logical shift left

`LSR Rd, Rn,#immed; Rd = Rn >> immed`
`LSR Rd, Rn ; Rd = Rd >> Rn`
`LSR.W Rd, Rn, Rm ; Rd = Rn >> Rm`
	Logical shift right

`ROR Rd, Rn ; Rd rot by Rn` `ROR.W Rd, Rn, Rm ; Rd = Rn rot by Rm`	Rotate right
`RRX.W Rd, Rn ; {C, Rd} = {Rn, C}`	Rotate right extended

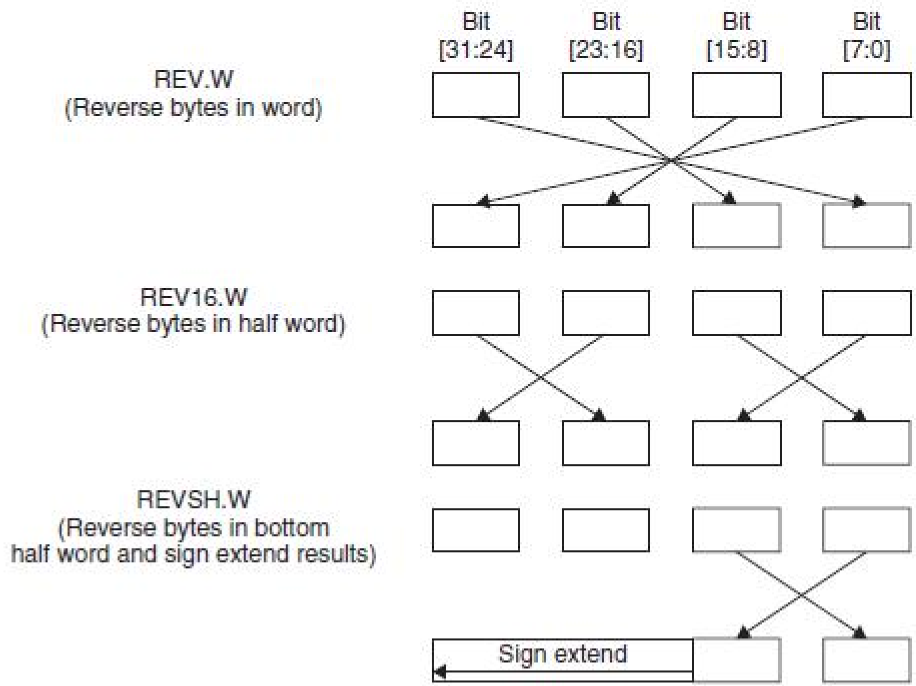


Sign Extend Instructions

Instruction	Operation
`SXTB.W Rd, Rm ; Rd = signext(Rn[7:0])`	Sign extend byte data into word
`SXTH.W Rd, Rm ; Rd = signext(Rn[15:0])`	Sign extend half word data into word

Data Reverse Ordering Instructions

Instruction	Operation
`REV.W Rd, Rn ; Rd = rev(Rn)`	Reverse bytes in word
`REV16.W <Rd>, <Rn> ; Rd = rev16(Rn)`	Reverse bytes in each half word
`REVSH.W <Rd>, <Rn> ; Rd = revsh(Rn)`	Reverse bytes in bottom half word and sign extend the result



ARM CortexM3 – Programmers view

Bit Field Processing and Manipulation Instructions

Instruction	Operation
BFC.W Rd, Rn, #<width>	Clear bit field within a register
BFI.W Rd, Rn, #<lsb>, #<width>	Insert bit field to a register
CLZ.W Rd, Rn	Count leading zero
RBIT.W Rd, Rn	Reverse bit order in register
SBFX.W Rd, Rn, #<lsb>, #<width>	Copy bit field from source and sign extend it
UBFX.W Rd, Rn, #<lsb>, #<width>	Copy bit field from source register

ARM CortexM3 – C to Assembly examples

```
i = 5;
while (i != 0 ){
func1(); ; call a function
i--;
}
```

This can be compiled into:

```
      MOV  R0, #5         ; Set loop counter
loop1 CBZ  R0, loop1exit  ; if loop counter = 0 then exit the lo
      BL   func1          ; call a function
      SUB  R0, #1         ; loop counter decrement
      B    loop1          ; next loop
loop1exit
```

ARM CortexM3 – C to Assembly examples

```
if (R1<R2) then
   R2=R2-R1
   R2=R2/2
else
   R1=R1-R2
   R1=R1/2
```

In assembly:

```
CMP       R1, R2         ; If R1 < R2 (less then)
ITTEE     LT             ; then execute instruction 1 and 2
                         ; (indicated by T)
                         ; else execute instruction 3 and 4
                         ; (indicated by E)
SUBLT.W   R2,R1          ; 1st instruction
LSRLT.W   R2,#1          ; 2nd instruction
SUBGE.W   R1,R2          ; 3rd instruction (notice the GE is
                         ; opposite of LT)
LSRGE.W   R1,#1          ; 4th instruction
```

Logical Break

Introduction to Image Processing

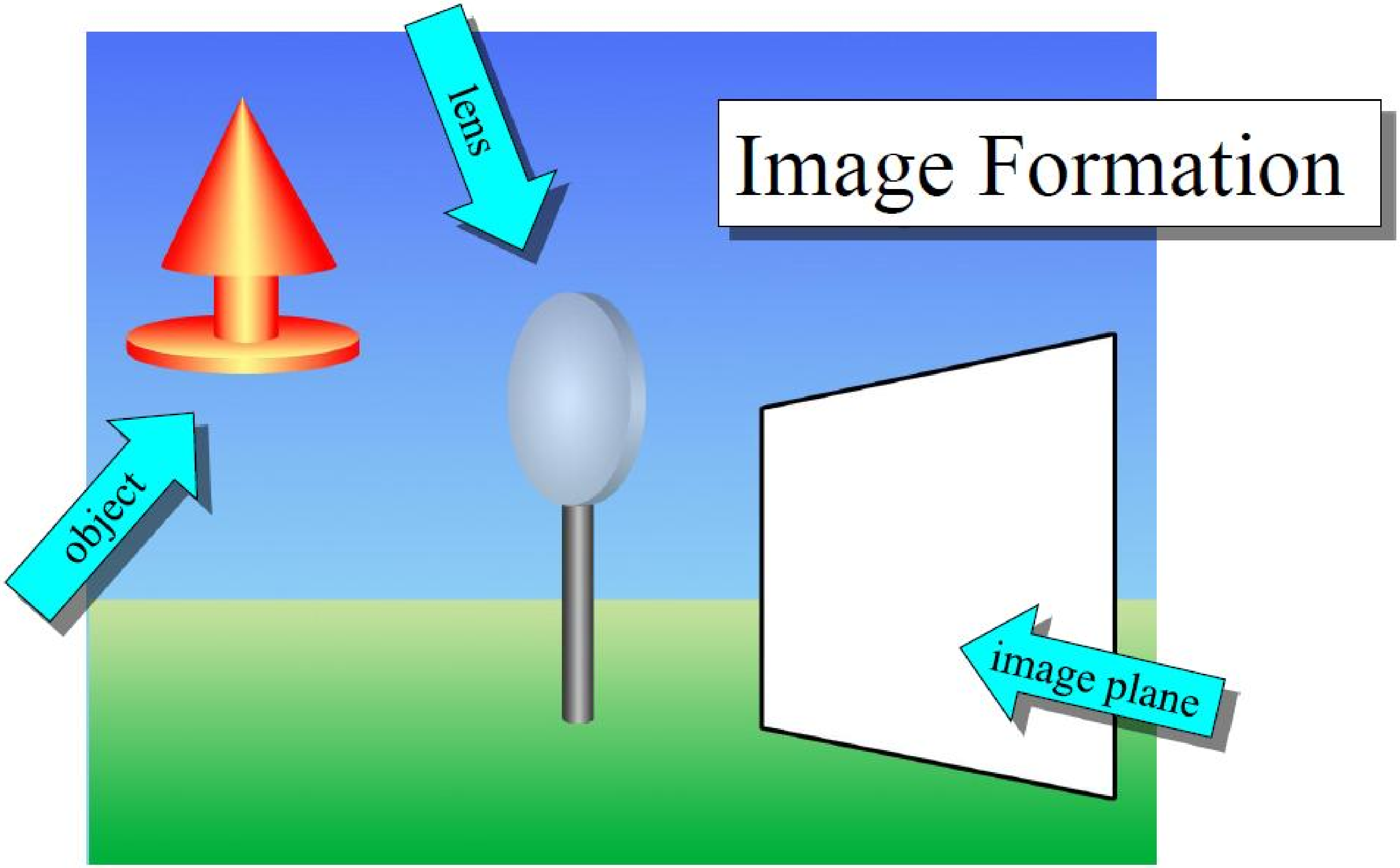
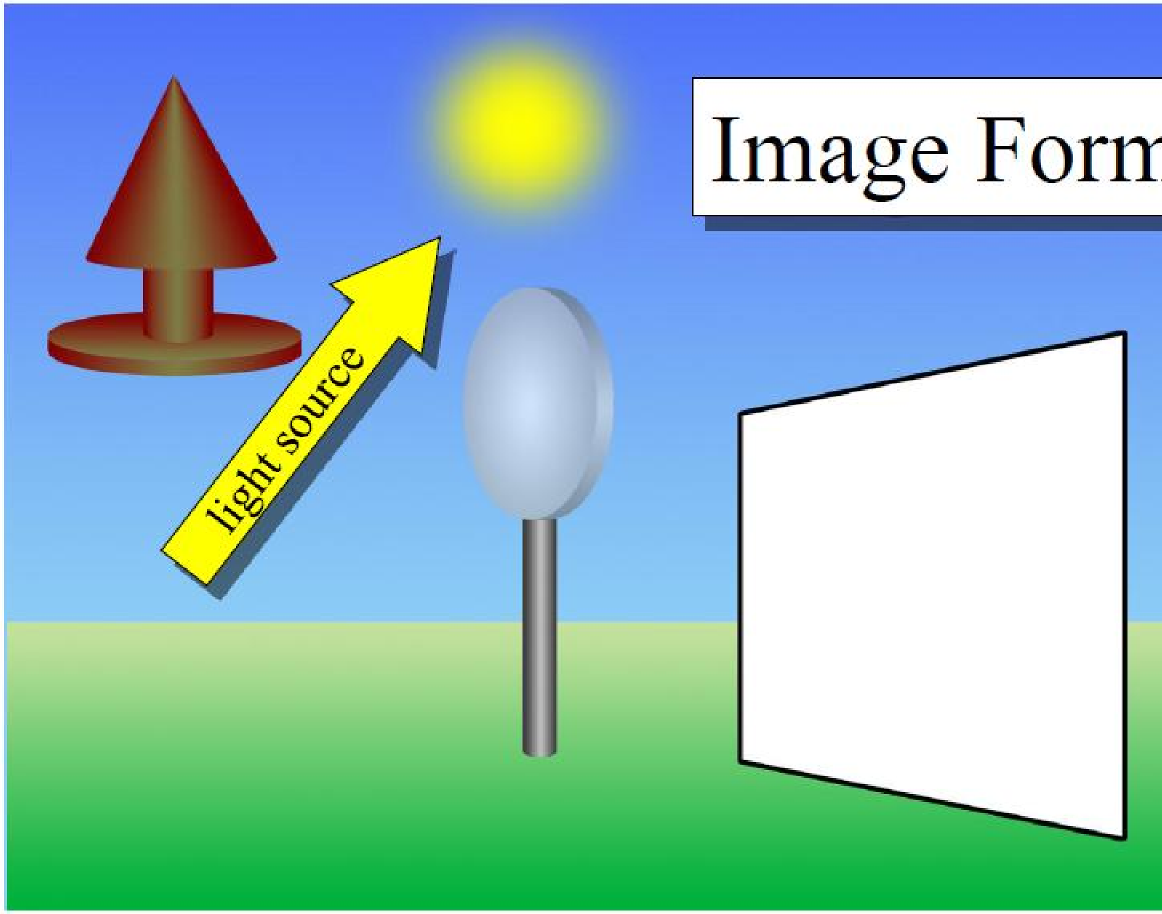


Image Formation



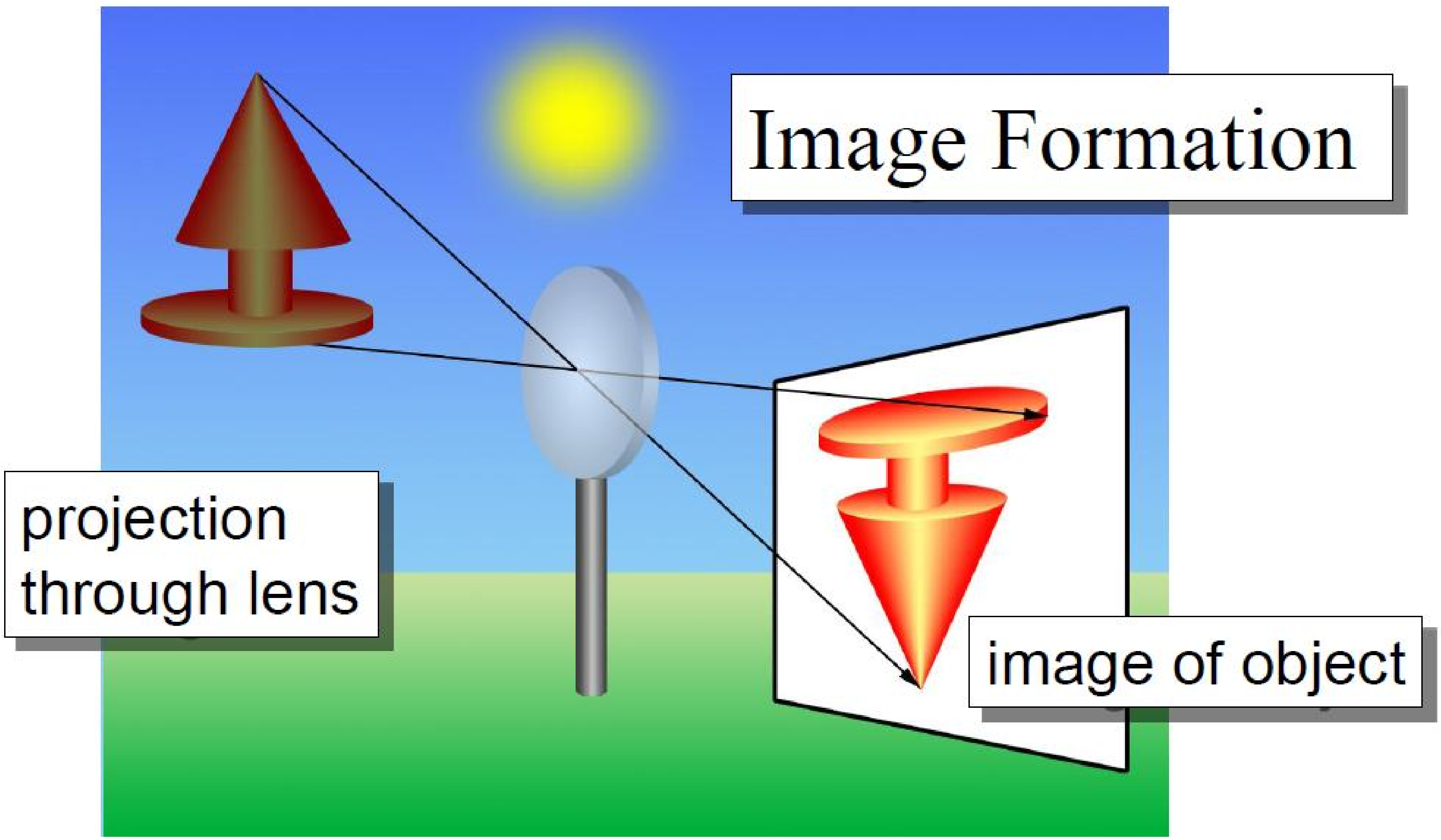


Image Formation
projection
through lens
image of object

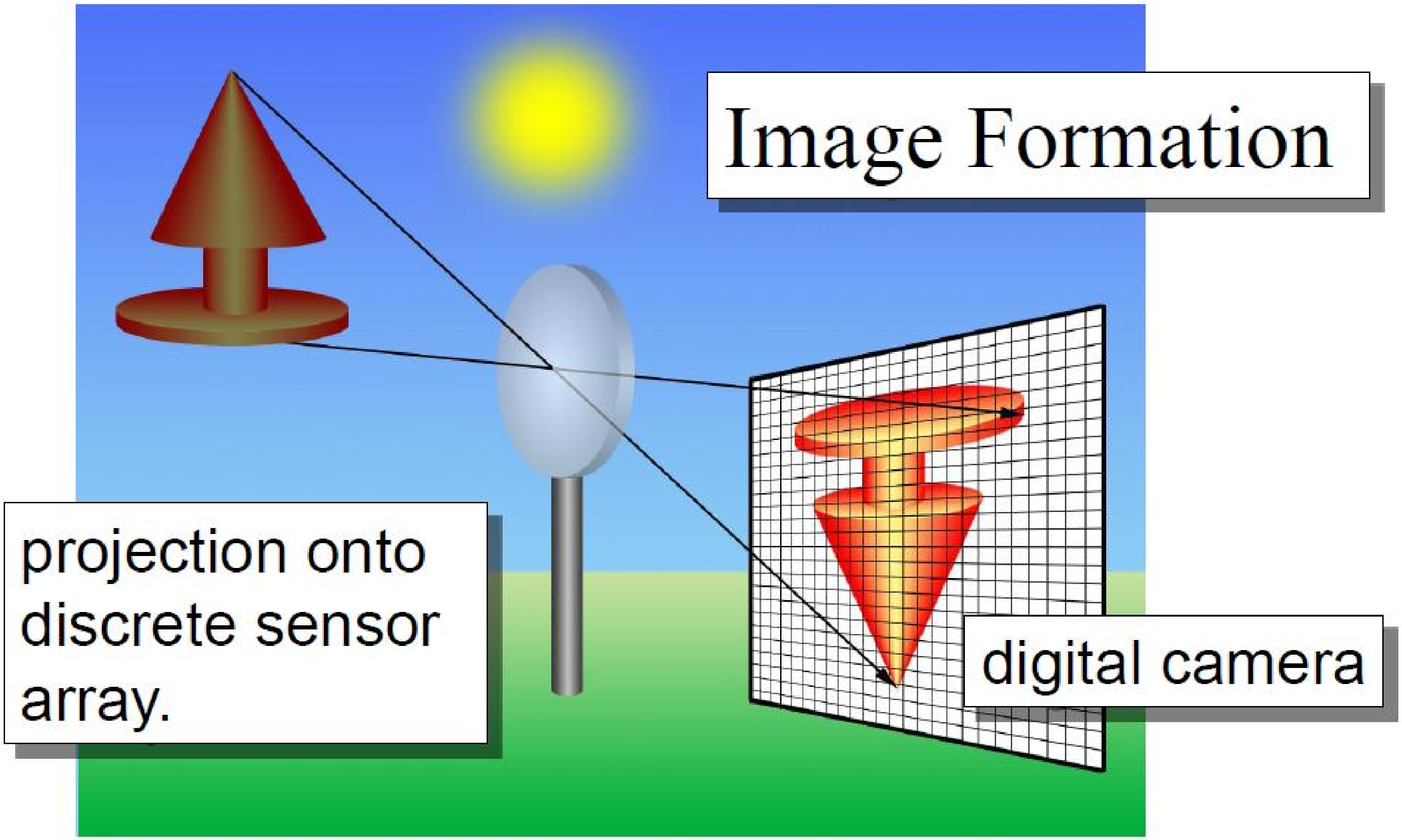


Image Formation
projection onto discrete sensor array.
digital camera

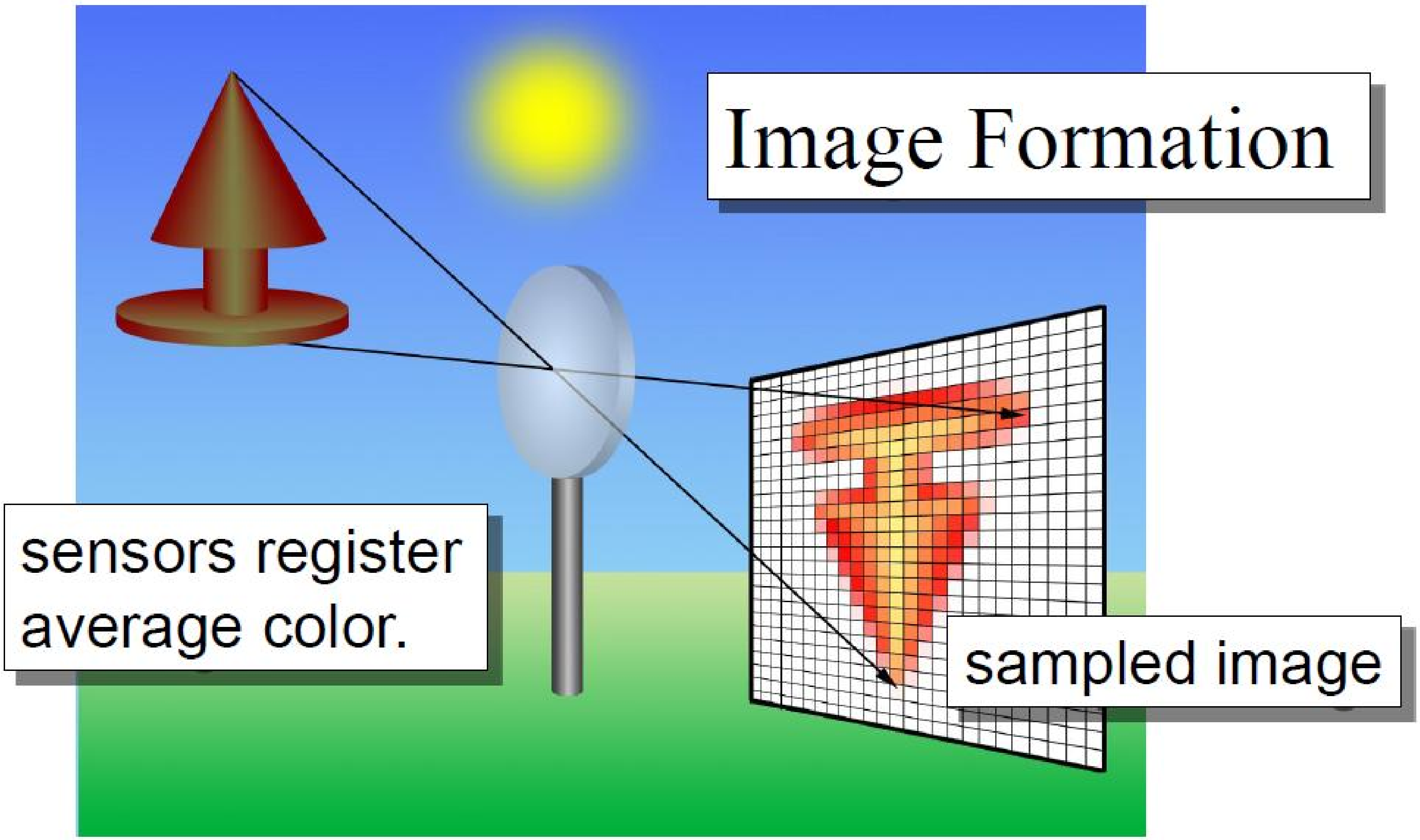


Image Formation
sensors register average color.
sampled image

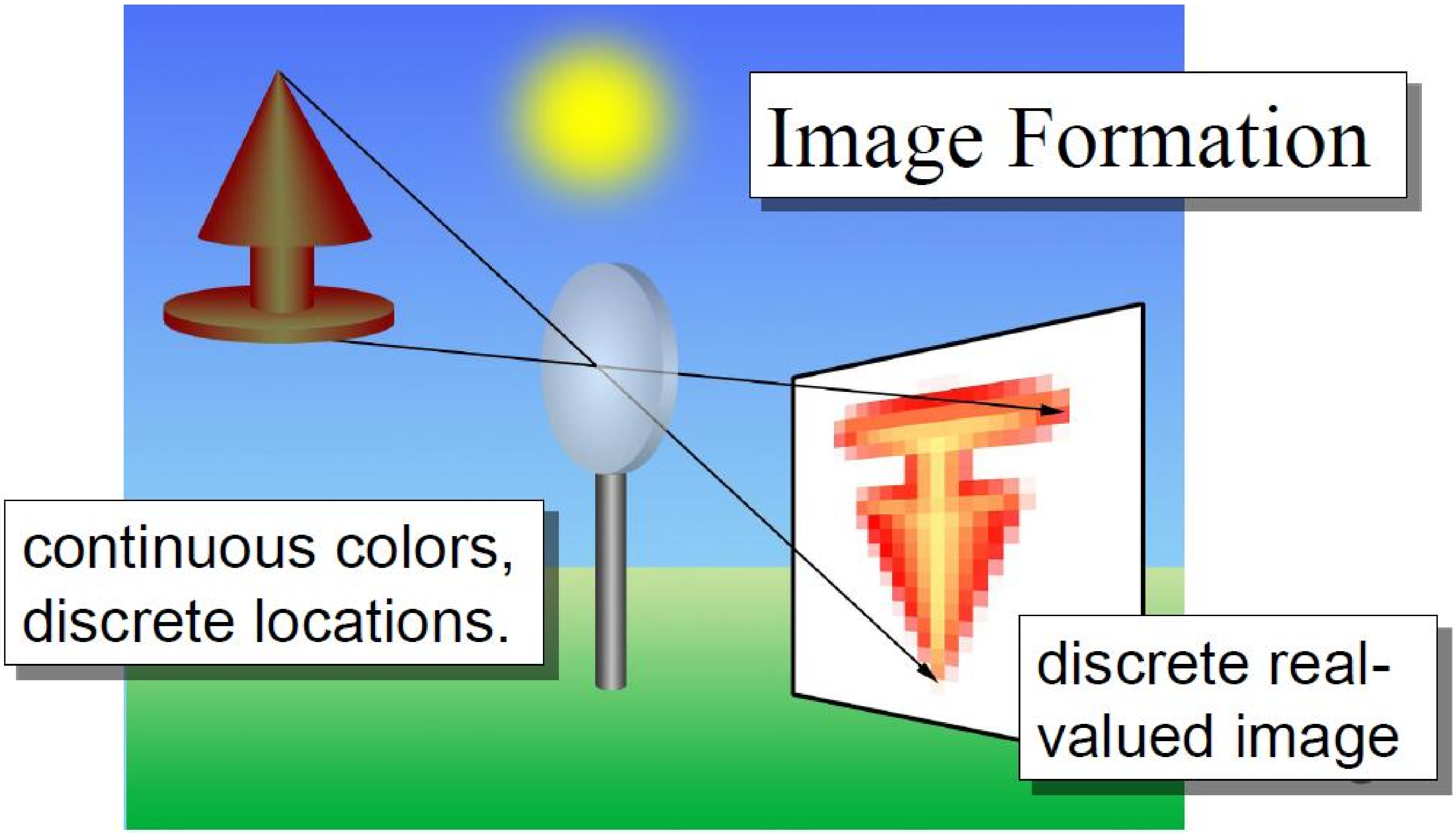
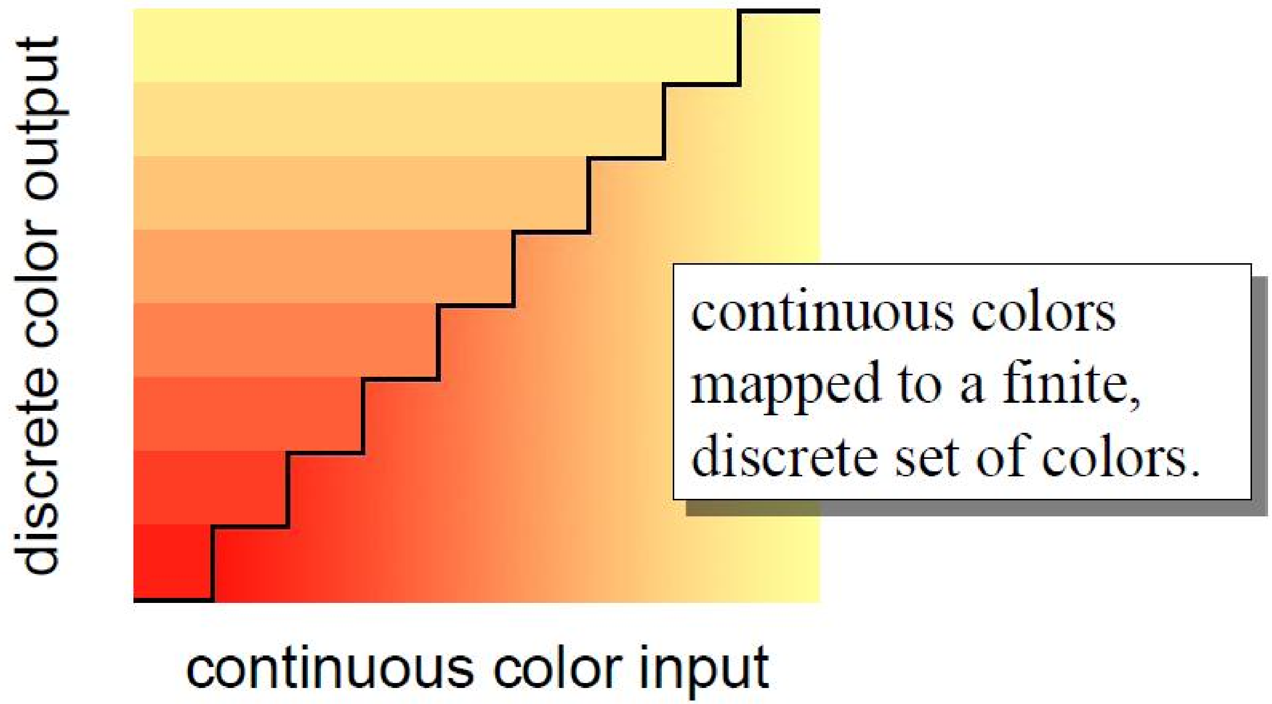
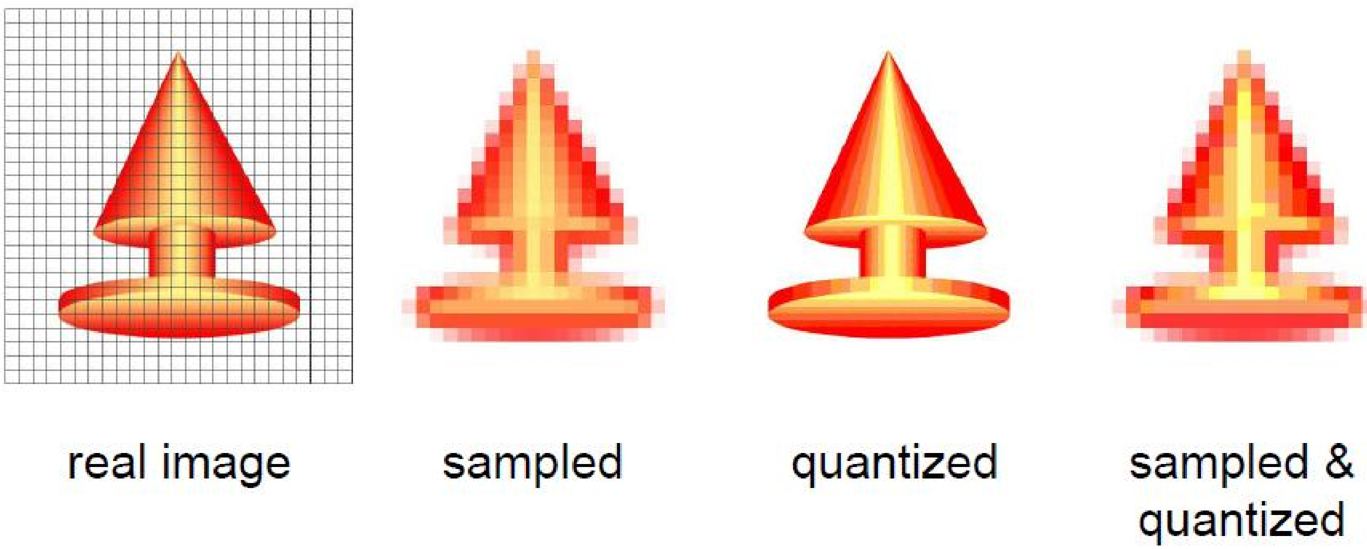


Image Formation
continuous colors,
discrete locations.
discrete real-
valued image

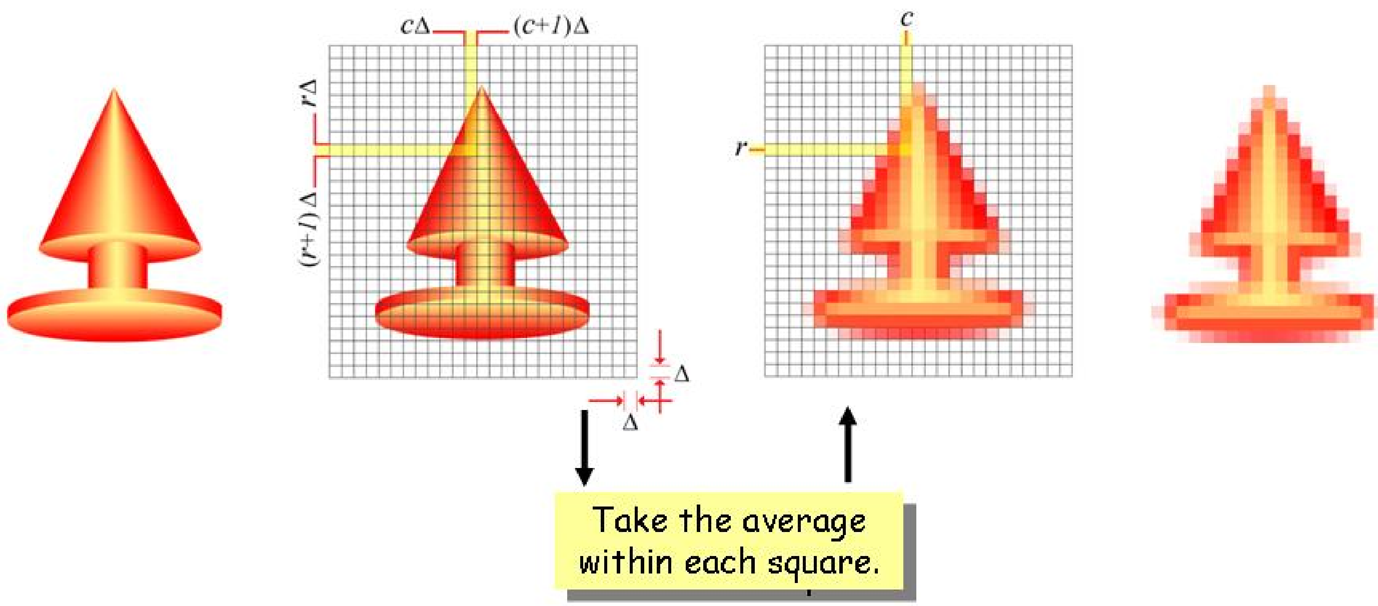
Digital Image Formation: Quantization



Sampling and Quantization



Pixelization :-



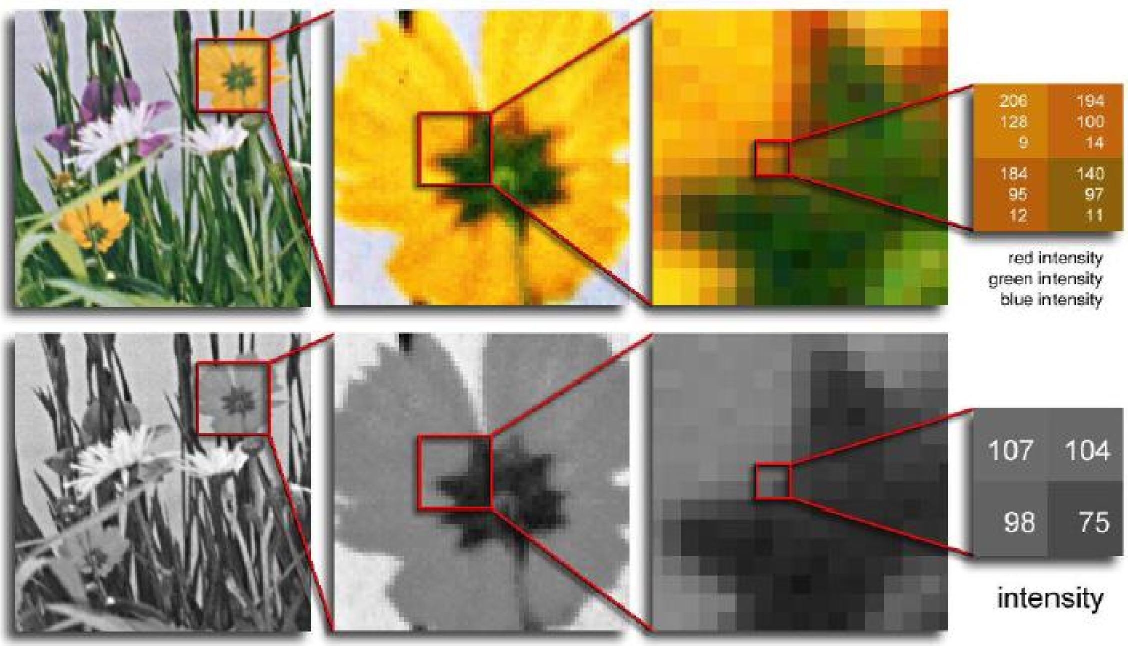
Pixelization :-

Digital Image

Color images have 3 values per pixel; monochrome images have 1 value per pixel.

a grid of squares, each of which contains a single color

each square is called a pixel (for *picture element*)



ARM CortexM3 – Programmers view and Development Environment

Introduction to Image Processing

Acknowledgements

1. ARM Assembler Reference :: www.arm.com

2. The Definitive Guide To ARM Cortex-M3 :: Joseph Yiu

3. http://www.archive.org/details/Lectures_on_Image_Processing :: Richard Alan Peters II :: Dept of Electrical Engg & CS :: Vanderbilt University School of Engineering.

   (a) EECE253_01_Intro.pdf

   (b) EECE253_03_PointProcessing.pdf

   (c) EECE253_10_PixelizationQuantization.pdf

4. YCbCr to RGB Considerations :: YCbCr_Intersil_AppNote.pdf

5. Embedded System Design - A Unified Hardware / Software Introduction :: Ch 7