Hochschule Heilbronn
Heilbronn University
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

Chapter 2: Stellaris® family of microcontrollers

Stellaris® Cortex™-M3 - Microcontroller Family

Texas Instruments

Texas Instruments - University Program

Heilbronn University, Campus Künzelsau

Prof. Dr.-Ing. Ralf Gessler

Rev. 1.0

Content

Hochschule Heilbronn
Heilbronn University
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

- Chapter 2: Stellaris® family of microcontrollers
  - 2.1 Overview MCU
  - 2.2 Cortex™-M3: Processor and Peripherals
  - 2.3 Cortex™-M3: Programmer model (ISA)
  - 2.4 Peripherals
- **Topics:** ISA, machine code, assembler, Cortex™-M3 instructions, addressing modes, core registers

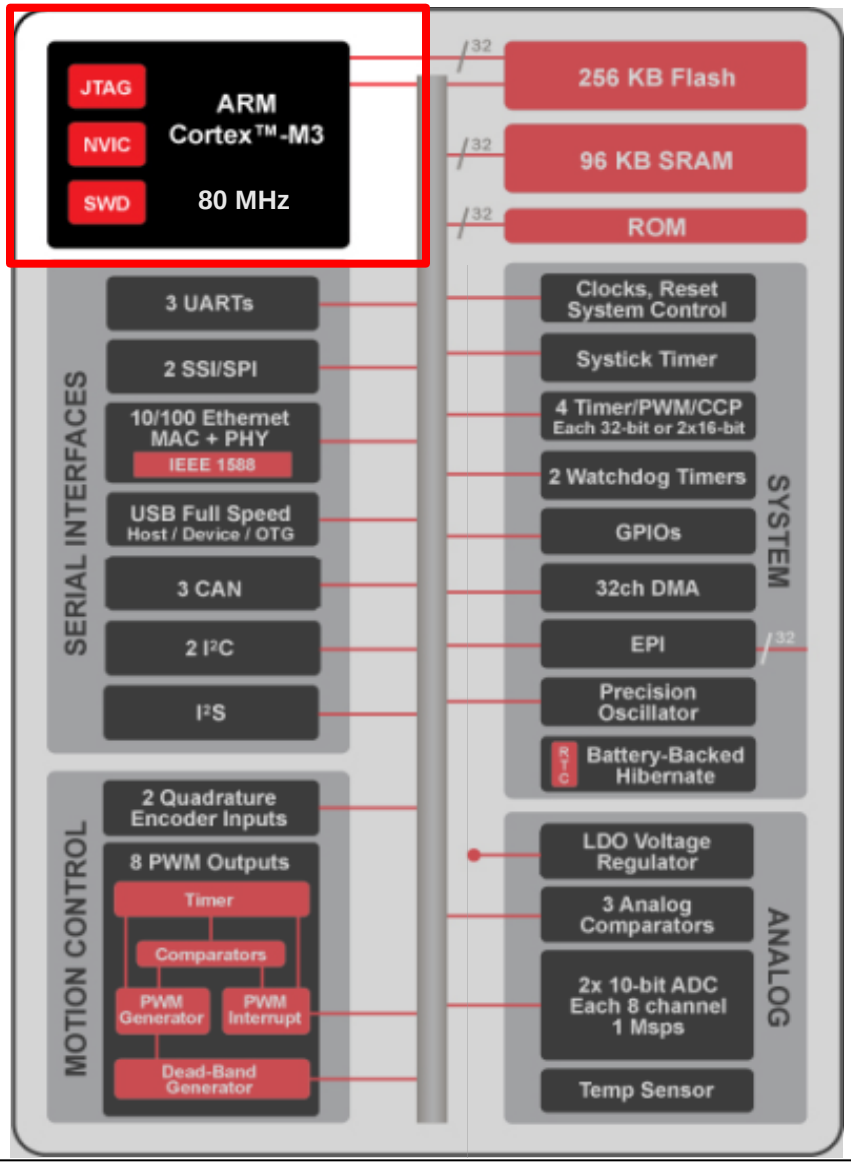
Learning Objectives

- The chapter describes the basics of the Cortex™-M3 Instruction Set Architecture (ISA).
- Its useful for a better understanding of the Cortex™ -M3 processor, code optimisation and debugging.
- Stellaris® family is optimised for C language.
- Therefore, the software projects are written only in the C language!
- Structure and questions:
  - What are ISA in general?
  - How does Cortex™-M3 instructions look like?
  - What are the addressing modes?
  - What are the Core registers?

General view

- Programming Model
  - Interface to the software developer
  - Digital circuit of Cortex™ -M3 is hidden
  - ISA is the only programming access to the processor
- Instruction Set Architecture (ISA)
  - Instruction Set
    - Example: MOV (move data)
  - Addressing Modes
    - Example: MOV Rd, #imm16
  - Register
    - Example: PC (Program Counter)
  - Data types
    - Example: 32-bit words
  - Data storage
    - Example: Stack
  - Special Features
    - Example: Processor Modes and Levels
  - …

Stellaris® Family Overview



Instruction Set Summary

- Machine code:
  - Bit sequence
  - Processor interprets these bits as instructions controlling the program flow
  - Unsuitable for programming
- Assembler
  - Machine oriented language
  - Programming with expressive abbreviations (Mnemonics)
  - Mnemonics are used in the introduced instruction set

Instruction Set Summary (cont.)

- Instructions supported by the Cortex-M3 processor
  - Memory Access Instructions
  - General Data Processing Instructions
  - Multiply and Divide Instructions
  - Saturating Instructions
  - Bit field Instructions
  - Branch and Control Instructions
  - Miscellaneous Instructions

Agenda

- Angle brackets, <>
  - Enclose alternative forms of the operand
- Braces, {}
  - Enclose optional operands
- Operands column
  - Is not exhaustive.
- Op2
  - Is a flexible second operand that can be either a register or a constant
- Most instructions
  - Can use an optional condition code suffix.

Agenda (cont.)

- cond
  - Is an optional condition code.
- Rd
  - Is the destination register. If Rd is omitted, the destination register is Rn.
- Rn
  - Is the register holding the first operand.
- label
  - Is a PC-relative expression
- S
  - Is an optional suffix. If S is specified, the condition code flags are updated on the result of the operation
- imm12
  - Is any value in the range 0-4095

Memory Access

Mnemonic	Brief Description
ADR	Load PC-relative address
CLREX	Clear exclusive
LDM{mode}	Load multiple registers
LDR{type}	Load register using immediate offset
LDR{type}	Load register using register offset
LDR{type}T	Load register with unprivileged access
LDR{type}	Load register using PC-relative address
LDRD	Load register using PC-relative address (two words)
LDREX{type}	Load register exclusive
POP	Pop registers from stack
PUSH	Push registers onto stack
STM{mode}	Store multiple registers
STR{type}	Store register using immediate offset
STR{type}	Store register using register offset
STR{type}T	Store register with unprivileged access
STREX{type}	Store register exclusive

General Data Processing

Mnemonic	Brief Description
ADC	Add with carry
ADD	Add
ADDW	Add
AND	Logical AND
ASR	Arithmetic shift right
BIC	Bit clear
CLZ	Count leading zeros
CMN	Compare negative
CMP	Compare
EOR	Exclusive OR
LSL	Logical shift left
LSR	Logical shift right
MOV	Move
MOVT	Move top
MOVW	Move 16-bit constant
MVN	Move NOT
ORN	Logical OR NOT
ORR	Logical OR
RBIT	Reverse bits
REV	Reverse byte order in a word
REV16	Reverse byte order in each halfword
REVSH	Reverse byte order in bottom halfword and sign extend
ROR	Rotate right
RRX	Rotate right with extend
RSB	Reverse subtract
SBC	Subtract with carry
SUB	Subtract
SUBW	Subtract
TEQ	Test equivalence
TST	Test

Multiply and Divide

Mnemonic	Brief Description
MLA	Multiply with accumulate, 32-bit result
MLS	Multiply and subtract, 32-bit result
MUL	Multiply, 32-bit result
SDIV	Signed divide
SMLAL	Signed multiply with accumulate (32x32+64), 64-bit result
SMULL	Signed multiply (32x32), 64-bit result
UDIV	Unsigned divide
UMLAL	Unsigned multiply with accumulate (32x32+64), 64-bit result
UMULL	Unsigned multiply (32x32), 64-bit result

Saturating

Mnemonic	Brief Description
SSAT	Signed saturate
USAT	Unsigned saturate

Bit field

Mnemonic	Brief Description
BFC	Bit field clear
BFI	Bit field insert
SBFX	Signed bit field extract
SXTB	Sign extend a byte
SXTH	Sign extend a halfword
UBFX	Unsigned bit field extract
UXTB	Zero extend a byte
UXTH	Zero extend a halfword

Branch and Control

Mnemonic	Brief Description
B	Branch
BL	Branch with link
BLX	Branch indirect with link
BX	Branch indirect
CBNZ	Compare and branch if non-zero
CBZ	Compare and branch if zero
IT	If-Then
TBB	Table branch byte
TBH	Table branch halfword

Miscellaneous

Mnemonic	Brief Description
BKPT	Breakpoint
CPSID	Change processor state, disable interrupts
CPSIE	Change processor state, enable interrupts
DMB	Data memory barrier
DSB	Data synchronization barrier
ISB	Instruction synchronization barrier
MRS	Move from special register to register
MSR	Move from register to special register
NOP	No operation
SEV	Send event
SVC	Supervisor call
WFE	Wait for event
WFI	Wait for interrupt

Addressing Modes

- Information about using the instructions:
  - Indexed Addressing
  - Operands
  - Restrictions When Using the PC or SP
  - Flexible Second Operand
  - Shift Operations
  - Address Alignment
  - PC-Relative Expressions
  - Conditional Execution
  - Instruction Width Selection

Addressing Modes (cont.)

- Offset addressing
  - The offset value is added to or subtracted from the address obtained from the register Rn. The result is used as the address for the memory access. The register Rn is unaltered. The assembly language syntax for this mode is:
    - [Rn, #offset]
- Pre-indexed addressing
  - The offset value is added to or subtracted from the address obtained from the register Rn. The result is used as the address for the memory access and written back into the register Rn. The assembly language syntax for this mode is:
    - [Rn, #offset]!

Addressing Modes (cont.)

- Post-indexed addressing
  - The address obtained from the register Rn is used as the address for the memory access. The offset value is added to or subtracted from the address, and written back into the register Rn. The assembly language syntax for this mode is: [Rn], #offset
- Operands
  - Can be
    - ARM Cortex-M3 register
    - Constant
    - or another instruction-specific parameter
  - Instructions act on the operands and often store the result in a destination register. When there is a destination register in the instruction, it is usually specified before the operands.
- Restrictions When Using the PC or SP
  - Many instructions have restrictions on whether you can use the Program Counter (PC) or Stack Pointer (SP) for the operands or destination register.

Addressing Modes (cont.)

- Flexible Second Operand
  - Many general data processing instructions have a flexible second operand. This is shown as Operand2 in the descriptions of the syntax of each instruction.
  - Operand2 can be a constant or a register with optional shift.
  - Constant
    - You specify an Operand2 constant in the form:
      - #constant
    - Where constant can be (X and Y are hexadecimal digits)
  - Register With Optional Shift

Addressing Modes (cont.)

- Register With Optional Shift (cont.)
  - You specify an Operand2 register in the form:
    - Rm {,shift}
  - "Where:"
  - Rm
    - Is the register holding the data for the second operand.
  - Shift
    - Is an optional shift to be applied to Rm. It can be one of:
    - ASR #n
    - Arithmetic shift right n bits, 1 ≤ n ≤ 32.
    - LSL #n
    - Logical shift left n bits, 1 ≤ n ≤ 31.
    - LSR #n
    - Logical shift right n bits, 1 ≤ n ≤ 32.
    - ROR #n
    - Rotate right n bits, 1 ≤ n ≤ 31.
    - RRX
    - Rotate right one bit, with extend.

Addressing Modes (cont.)

- Shift Operations
  - Register shift operations move the bits in a register left or right by a specified number of bits, the shift length. Register shift can be performed:
    - Directly by the instructions ASR, LSR, LSL, ROR, and RRX, and the result is written to a destination register.
    - During the calculation of Operand2 by the instructions that specify the second operand as a register with shift.
- Address Alignment
  - An aligned access is an operation where a word-aligned address is used for a word, dual word, or multiple word access, or where a halfword-aligned address is used for a halfword access. Byte accesses are always aligned.
  - The Cortex-M3 processor supports unaligned access only for the following instructions:
    - LDR, LDRT
    - LDRH, LDRHT
    - LDRSH, LDRSHT
    - STR, STRT
    - STRH, STRHT

Addressing Modes (cont.)

- PC-Relative Expressions
  - A PC-relative expression or label is a symbol that represents the address of an instruction or literal data. It is represented in the instruction as the PC value plus or minus a numeric offset.
  - The assembler calculates the required offset from the label and the address of the current instruction. If offset is too big, assembler produces an error.
- Conditional Execution
  - Most data processing instructions can optionally update the condition flags in the Application Program Status Register (APSR) register according to the result of the operation.
  - You can execute an instruction conditionally, based on the condition flags set in another instruction, either immediately after the instruction that updated the flags, or after any number of intervening instructions that have not updated the flags.
  - Conditional instructions, except for conditional branches, must be inside an If-Then instruction block (see IT instruction).

Addressing Modes (cont.)

- Conditional Execution (cont.)
  - Condition Code Suffixes
    - The instructions that can be conditional have an optional condition code, shown in syntax descriptions as {cond}.
    - Conditional execution requires a preceding IT instruction.
    - An instruction with a condition code is only executed if the condition code flags in APSR meet the specified condition.

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N != V	Less than or equal, signed ≤
AL	Can have any value	Always. This is the default when no suffix is specified.

Addressing Modes (cont.)

- Conditional Execution (cont.)
  - Example Absolute Value
    - MOVS R0, R1 ; R0 = R1, setting flags.
    - IT MI ; IT instruction for the negative condition.
    - RSBMI R0, R1, #0 ; If negative, R0 = -R1.
  - Example Compare and Update Value
    - CMP R0, R1 ; Compare R0 and R1, setting flags
    - ITT GT ; IT instruction for the two GT conditions
    - CMPGT R2, R3 ; If 'greater than',
; compare R2 and R3, setting flags
    - MOVGT R4, R5 ; If still 'greater than', do R4 = R5

Addressing Modes (cont.)

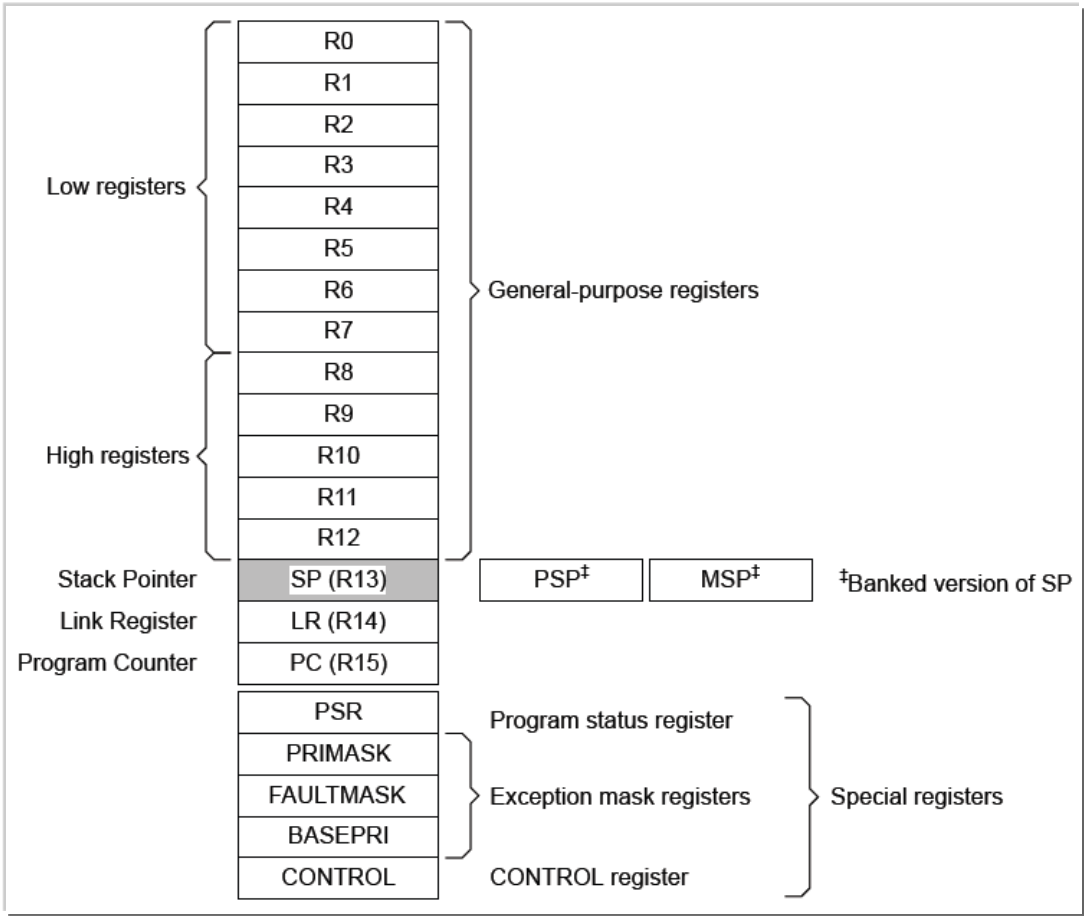
- Instruction Width Selection
  - There are many instructions that can generate either a 16-bit encoding or a 32-bit encoding depending on the operands and destination register specified. For some of these instructions, you can force a specific instruction size by using an instruction width suffix.
  - The .W suffix forces a 32-bit instruction encoding.
  - The .N suffix forces a 16-bit instruction encoding.

Examples: Assembler

- MOV
  - Move
  - Syntax
    - MOV{S}{cond} Rd, Operand2
    - MOV{cond} Rd, #imm16
  - MOVS R11, #0x000B ; Write value of 0x000B to R11, flags get updated.
  - MOV R1, #0xFA05 ; Write value of 0xFA05 to R1, flags are not updated.
  - MOVS R10, R12 ; Write value in R12 to R10, flags get updated.
  - MOV R3, #23 ; Write value of 23 to R3.
  - MOV R8, SP ; Write value of stack pointer to R8.
- IT
  - If-Then
  - ISA Cortex™-M3 supports high-level languages like C

Core Register Map

- Cortex™-M3 Register Set



Core Register Map (cont.)

- General-Purpose Registers
  - Rn registers are 32-bit general-purpose registers for data operations.
- Stack Pointer (SP)
  - Depends on ASP bit in the Control Register (CONTROL)
    - ASP bit is set: Process Stack Pointer (PSP)
    - ASP bit is clear: Main Stack Pointer (MSP)
- Link Register (LR)
  - Stores the return information for subroutines, function calls, and exceptions.
- Program Counter (PC)
  - Contains the current program address

Core Register Map (cont.)

- The Program Status Register (PSR) has three functions, and the register bits are assigned to the different functions:
  - Application Program Status Register (APSR), bits 31:27,
  - Execution Program Status Register (EPSR), bits 26:24, 15:10
  - Interrupt Program Status Register (IPSR), bits 5:0
  - Note: This register is also referred to as xPSR.
- APSR contains the current state of the condition flags from previous instruction executions:
  - N: Set to 1 when the result of the operation was negative; cleared to 0 otherwise.
  - Z: Set to 1 when the result of the operation was zero; cleared to 0 otherwise.
  - C: Set to 1 when the operation resulted in a carry; cleared to 0 otherwise.
  - V: Set to 1 when the operation caused overflow; cleared to 0 otherwise.

Core Register Map (cont.)

- Priority Mask Register (PRIMASK)
  - Prevents activation of all exceptions with programmable priority.
- Fault Mask Register (FAULTMASK)
  - Prevents activation of all exceptions except for the Non-Maskable Interrupt (NMI).
- Base Priority Mask Register (BASEPRI)
  - Defines the minimum priority for exception processing.
- Control Register (CONTROL)
  - Controls the stack used and the privilege level for software execution when the processor is in Thread mode.
  - This register is only accessible in privileged mode.

Data Types

- The Cortex™ -M3 supports
  - 32-bit words
  - 16-bit half words
  - 8-bit bytes.
- The processor also supports 64-bit data transfer instructions.
- All instruction and data memory accesses are little endian.
- Note: Little Endian - byte-order: lower addresses have lower significance (least significant bits)

Stacks

- Stack
  - LIFO storage
  - for saving addresses, registers etc. from subroutine, interrupt calls
- The processor uses a full descending stack, meaning that the stack pointer indicates the last stacked item on the stack memory.
- When the processor pushes a new item onto the stack, it decrements the stack pointer and then writes the item to the new memory location.
- The processor implements two stacks: the main stack and the process stack, with independent copies of the stack pointer (SP)

Modes and Levels

- Software Execution
  - Processor Mode
  - Privilege Levels
- Cortex™-M3 has two modes of operation:
  - Thread mode
    - Used to execute application software.
      The processor enters Thread mode when it comes out of reset.
  - Handler mode
    - Used to handle exceptions.
      When the processor has finished exception processing, it returns to Thread mode.

Modes and Levels (cont.)

- Cortex™-M3 has two privilege levels:
  - Unprivileged
    - In this mode, software has the following restrictions:
    - Limited access to the MSR and MRS instructions and no use of the CPS instruction
    - No access to the system timer, NVIC, or system control block
    - Possibly restricted access to memory or peripherals
  - Privileged
    - In this mode, software can use all the instructions and has access to all resources.
- CONTROL register
  - In Thread mode, the CONTROL register controls whether software execution is privileged or unprivileged. In Handler mode, software execution is always privileged.
  - Only privileged software can write to the CONTROL register to change the privilege level for software execution in Thread mode.

Exceptions and Interrupts

- The Cortex™-M3 processor supports interrupts and system exceptions.
- The processor and the Nested Vectored Interrupt Controller (NVIC) prioritize and handle all exceptions.
- An exception changes the normal flow of software control.
- The processor uses Handler mode to handle all exceptions except for reset.
- Link: Chapter 2.2 “Cortex™-M3 Processor and Peripherals”

Example: Disassembly

- Basic instructions
  - for …
  - if … else …
- Link: Chapter 3.3 “C language: Introduction”

Questions and Exercises

Hochschule Heilbronn
Heilbronn University
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

1. Explain the expressions:
   Machine code, Assembler, Mnemonics.
2. What is ISA?
3. What are the modes and levels of Cortex™-M3?
4. Explain the Register Set of Cortex™-M3.

Summary and Outlook

Hochschule Heilbronn
Heilbronn University
ENGINEERING BUSINESS INFORMATICS
Campus Künzelsau
Reinhold-Würth-Hochschule

- Summary
  - From ISA in general to Cortex™-M3
  - From the Instruction set to Addressing modes
- Outlook/How to go on?
  - Chapter 3.3 “C language: Introduction” describes the design flow from C to Assembler.
  - Chapter 3.2 “Software development tools” shows the practical use of the Cortex™-M3 ISA.

References

- [1] Texas Instruments: *Data Sheet - Stellaris® LM3S1968 Microcontroller.* Chapter 2.3: “Programming Model”, Chapter 2.8: “Instruction Set Summary”, spms037f.pdf, 2011.
- [2] Texas Instruments: *Technical User’s Manual - Cortex-M3 Instruction Set.* spmu159.pdf, 2010.
- [3] Henri, G.; Texas Instruments: *MCU Training Module - ARM® Cortex™ M3/M4 cores*. EMEA, Oct. 7, 2010.