ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN-ĐIỆN TỬ
BỘ MÔN KỸ THUẬT ĐIỆN TỬ

Embedded System Design

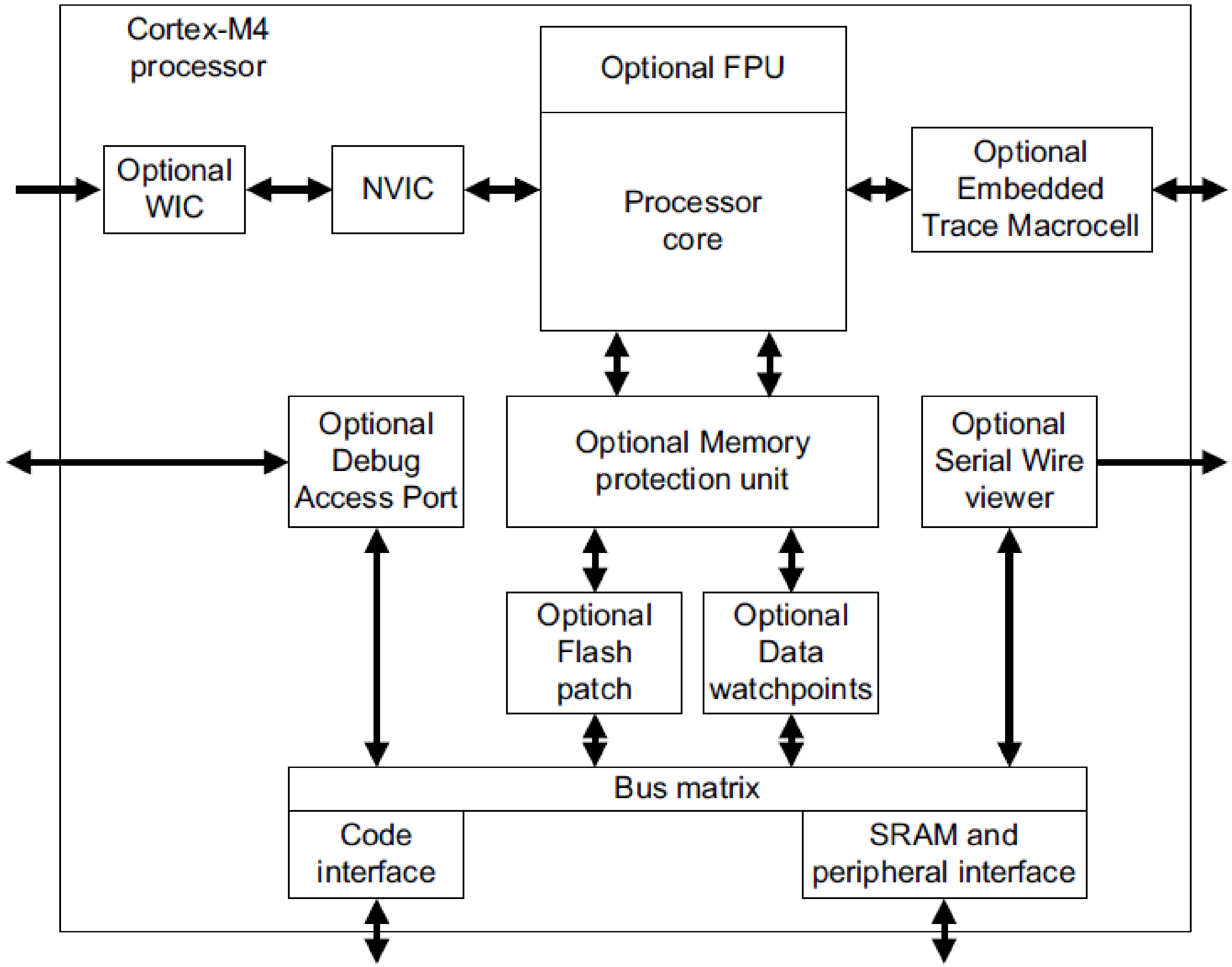
Chapter 2: Microcontroller Series (Part 2)

3. ARM Cortex-M4
4. ARM Cortex-M3 & M4 Microcontroller Series
5. ARM programming

3. ARM Cortex-M4

- Features
  - tight integration of system peripherals reduces area and development costs
  - Thumb instruction set combines high code density with 32-bit performance
  - **optional IEEE754-compliant single-precision FPU**
  - code-patch ability for ROM system updates
  - power control optimization of system components
  - integrated sleep modes for low power consumption
  - **DSP extension: Single cycle 16/32-bit MAC, single cycle dual 16-bit MAC, 8/16-bit SIMD arithmetic**

3. ARM Cortex-M4 - Architecture



3. ARM Cortex-M4 – Peripherals

- **Nested Vectored Interrupt Controller**
  - An interrupt controller that supports low latency interrupt processing.
- **System Control Block**
  - Provides system implementation information and system control, including configuration, control, and reporting of system exceptions.
- **System timer**
  - The system timer, SysTick, is a 24-bit count-down timer. Use this as a Real Time Operating System (RTOS) tick timer or as a simple counter.
- **Memory Protection Unit**
  - *defining the* memory attributes for different memory regions. It provides up to eight different regions, and an optional predefined background region.
- **Floating-point Unit**
  - The *Floating-Point Unit (FPU) provides IEEE754-compliant operations on* single-precision, 32-bit, floating-point values.

3. ARM Cortex-M4 – Instruction Set

- Extra instructions for floating point operations

VABS.F32	Sd, Sm	Floating-point Absolute
VADD.F32	{Sd,} Sn, Sm	Floating-point Add
VCMP.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero
VCMPE.F32	Sd, <Sm | #0.0>	Compare two floating-point registers, or one floating-point register and zero with Invalid Operation check
VCVT.S32.F32	Sd, Sm	Convert between floating-point and integer
VCVT.S16.F32	Sd, Sd, #fbits	Convert between floating-point and fixed point
VCVTR.S32.F32	Sd, Sm	Convert between floating-point and integer with rounding

3. ARM Cortex-M4 – Instruction Set

VCVT<B|H>.F32.F16	Sd, Sm	Converts half-precision value to single-precision
VCVTT<B|T>.F32.F16	Sd, Sm	Converts single-precision register to half-precision
VDIV.F32	{Sd,} Sn, Sm	Floating-point Divide
VFMA.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Accumulate
VFNMA.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Accumulate
VFMS.F32	{Sd,} Sn, Sm	Floating-point Fused Multiply Subtract
VFNMS.F32	{Sd,} Sn, Sm	Floating-point Fused Negate Multiply Subtract
VLDM.F<32|64>	Rn{!}, list	Load Multiple extension registers
VLDR.F<32|64>	<Dd|Sd>, [Rn]	Load an extension register from memory
VLMA.F32	{Sd,} Sn, Sm	Floating-point Multiply Accumulate
VLMS.F32	{Sd,} Sn, Sm	Floating-point Multiply Subtract
VMOV.F32	Sd, #imm	Floating-point Move immediate

3. ARM Cortex-M4 – Instruction Set

VMOV	Sd, Sm	Floating-point Move register
VMOV	Sn, Rt	Copy ARM core register to single precision
VMOV	Sm, Sm1, Rt, Rt2	Copy 2 ARM core registers to 2 single precision
VMOV	Dd[x], Rt	Copy ARM core register to scalar
VMOV	Rt, Dn[x]	Copy scalar to ARM core register
VMRS	Rt, FPSCR	Move FPSCR to ARM core register or APSR
VMSR	FPSCR, Rt	Move to FPSCR from ARM Core register
VMUL.F32	{Sd,} Sn, Sm	Floating-point Multiply
VNEG.F32	Sd, Sm	Floating-point Negate
VNMLA.F32	Sd, Sn, Sm	Floating-point Multiply and Add
VNMLS.F32	Sd, Sn, Sm	Floating-point Multiply and Subtract
VNMUL	{Sd,} Sn, Sm	Floating-point Multiply

3. ARM Cortex-M4 – Instruction Set

VPOP	list	Pop extension registers
VPUSH	list	Push extension registers
VSQRT.F32	Sd, Sm	Calculates floating-point Square Root
VSTM	Rn{!}, list	Floating-point register Store Multiple
VSTR.F<32|64>	Sd, [Rn]	Stores an extension register to memory
VSUB.F<32|64>	{Sd,} Sn, Sm	Floating-point Subtract

3. ARM Cortex-M4 – Instruction Set

- Extra instructions for DSP

UADD16	{Rd,} Rn, Rm	Unsigned Add 16
UADD8	{Rd,} Rn, Rm	Unsigned Add 8
USAX	{Rd,} Rn, Rm	Unsigned Subtract and Add with Exchange
UHADD16	{Rd,} Rn, Rm	Unsigned Halving Add 16
UHADD8	{Rd,} Rn, Rm	Unsigned Halving Add 8
UHASX	{Rd,} Rn, Rm	Unsigned Halving Add and Subtract with Exchange
UHSAX	{Rd,} Rn, Rm	Unsigned Halving Subtract and Add with Exchange
UHSUB16	{Rd,} Rn, Rm	Unsigned Halving Subtract 16
UHSUB8	{Rd,} Rn, Rm	Unsigned Halving Subtract 8
UBFX	Rd, Rn, #lsb, #width	Unsigned Bit Field Extract
UDIV	{Rd,} Rn, Rm	Unsigned Divide

3. ARM Cortex-M4 – Instruction Set

- Extra instructions for DSP

UMAAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply Accumulate Accumulate Long (32 x 32 + 32 +32), 64-bit result
UMLAL	RdLo, RdHi, Rn, Rm	Unsigned Multiply with Accumulate (32 x 32 + 64), 64-bit result
UMULL	RdLo, RdHi, Rn, Rm	Unsigned Multiply (32 x 32), 64-bit result
UQADD16	{Rd,} Rn, Rm	Unsigned Saturating Add 16
UQADD8	{Rd,} Rn, Rm	Unsigned Saturating Add 8

3. ARM Cortex-M4 – Instruction Set

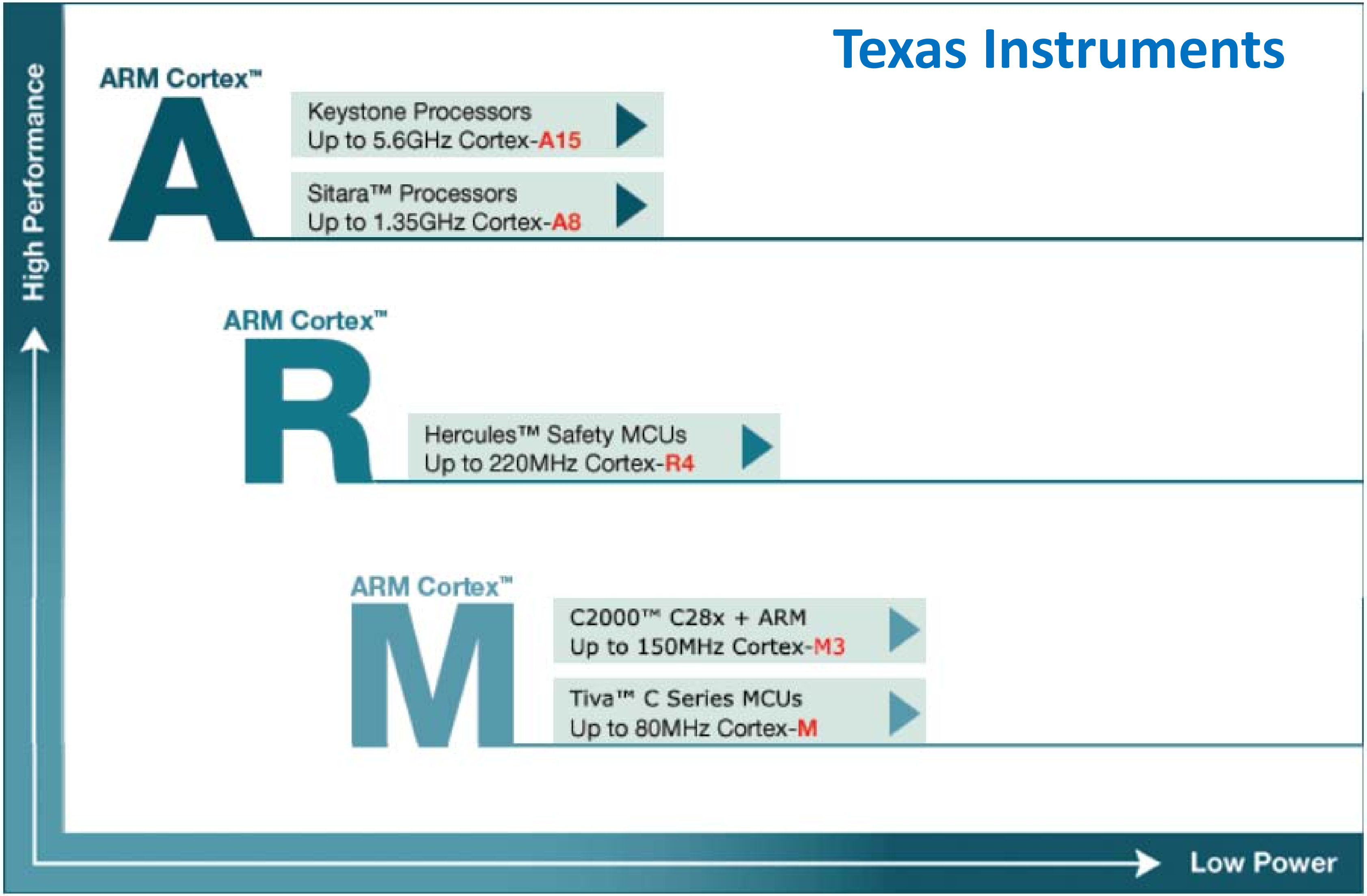
- Extra instructions for DSP

UQASX	{Rd,} Rn, Rm	Unsigned Saturating Add and Subtract with Exchange
UQSAX	{Rd,} Rn, Rm	Unsigned Saturating Subtract and Add with Exchange
UQSUB16	{Rd,} Rn, Rm	Unsigned Saturating Subtract 16
UQSUB8	{Rd,} Rn, Rm	Unsigned Saturating Subtract 8
USAD8	{Rd,} Rn, Rm	Unsigned Sum of Absolute Differences
USADA8	{Rd,} Rn, Rm, Ra	Unsigned Sum of Absolute Differences and Accumulate
USAT	Rd, #n, Rm {,shift #s}	Unsigned Saturate
USAT16	Rd, #n, Rm	Unsigned Saturate 16

4. ARM Cortex Microcontroller Series

ST Electronics

4. ARM Cortex Microcontroller Series



4. ARM Cortex Microcontroller Series

- **Stellaris® Roadmap**

Production | Sampling | Development

ARM Cortex-M3

LM3S9000
LM3S8000
LM3S6000

Fixed Point
ENET MAC & PHY
USB & CAN options

LM3S5000
LM3S3000

Fixed Point
USB H/D/OTG
CAN options

LM3S2000
LM3S1000
LM3S800

Fixed Point
General Purpose
CAN options

ARM Cortex-M4F Floating-Point

LM4F23x
LM4F13x

RTP Feb '13 (TMX Now)
- USB H/D/OTG + CAN
- 80 MHz
- 256K Flash / 32K SRAM
- Low-power hibernate
- 2 x 1 Msps 12-bit ADCs
- Motion control options

LM4F21x
LM4F12x
LM4F11x

RTP Feb '13 (TMX Now)
- 80 MHz
- 256K Flash / 32K SRAM
- Low-power hibernate
- 2 x 1 Msps 12-bit ADCs
- Up to 2 x CAN
- Motion control options

LM4F29x

TMS / RTP 2H13

Ethernet + USB + CAN
- 120 MHz
- 1MB Flash, 256KB SRAM
- 10/100 ENET MAC + PHY
- USB H/D/OTG w/FS PHY & HS ULPI
- Up to 2 x CAN
- Parallel Bus Interface (EPI)
- Crypto

LM4F29x

TMS / RTP 2H13

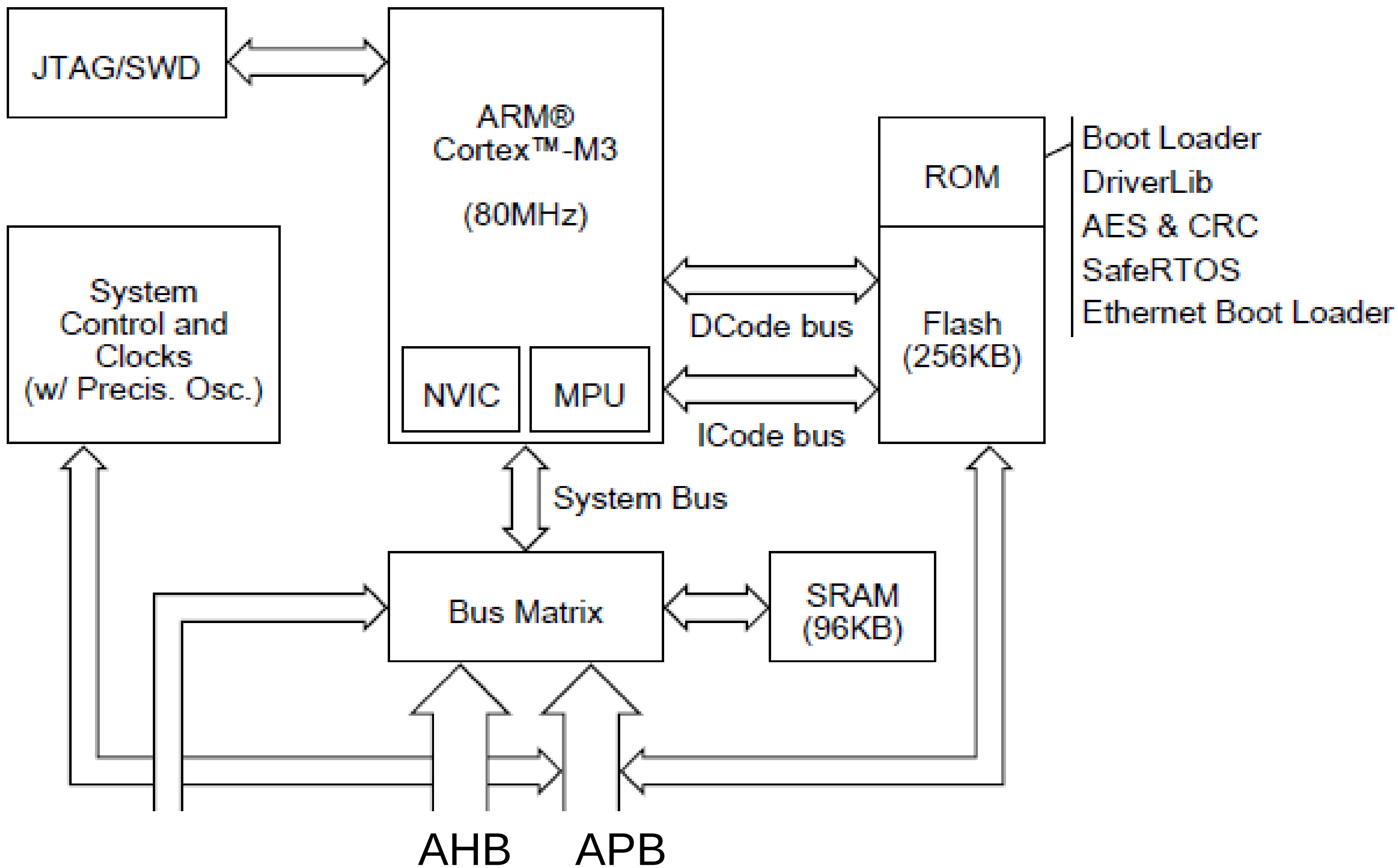
USB + CAN
- 120 MHz
- 1MB Flash, 256KB SRAM
- USB H/D/OTG w/FS PHY & HS ULPI
- Up to 2 x CAN
- Parallel Bus Interface (EPI)
- Crypto

4.1 ARM Cortex-M3 – LM3S9B96

- Stellaris LM3S9B96 microcontroller (Texas Instruments)
  - ARM Cortex-M3
  - 80MHz, 100 DMIPS
  - 256 KB Flash
  - 96KB SRAM
  - UART, SSI, I2C, I2S, CAN, Ethernet, USB
  - Timer, DMA, GPIO
  - PWM
  - ADC
  - JTAG, ARM Serial Wire Debug

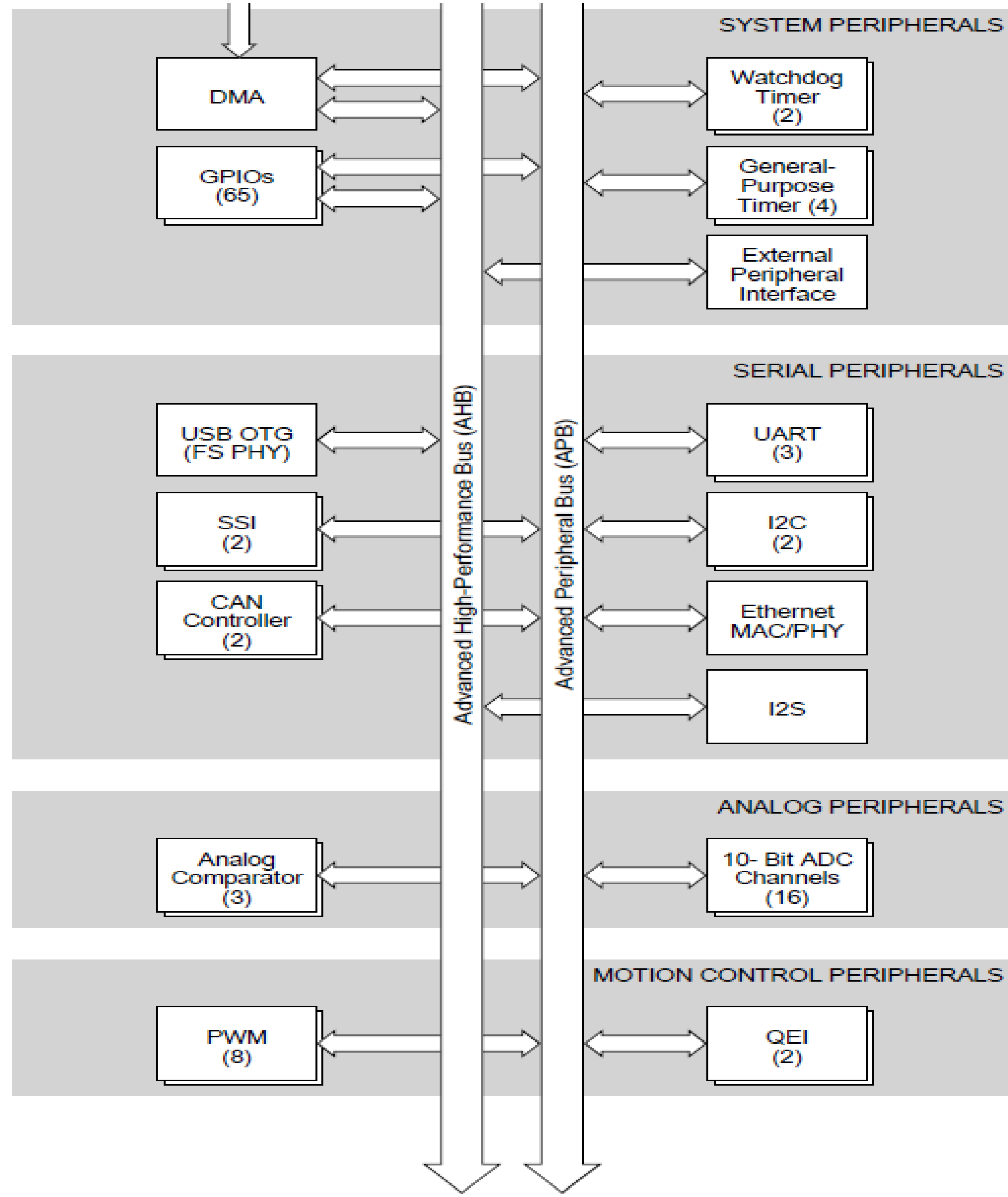
4.1 ARM Cortex-M3 – LM3S9B96

- Block diagram:
  - 2 on-chip buses: AHB, APB



4.1 ARM Cortex-M3 – LM3S9B96

- Target applications
  - Gaming equipment
  - Network appliances
  - Home and commercial site monitoring
  - Motion control
  - Medical instruments
  - Remote monitoring
  - Test equipment
  - Fire and security
  - Lighting control
  - Transportation



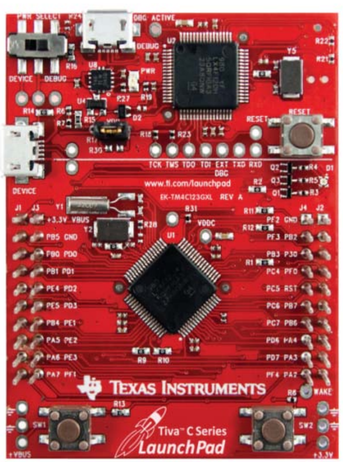
4.2 ARM Cortex-M4 – TM4F

- Features:
  - ARM Cortex-M4F core CPU speed up to 80 MHz with floating point
  - Up to 256-KB Flash
  - Up to 32-KB single-cycle SRAM
  - Two high-speed 12-bit ADCs up to 1MSPS
  - Up to two CAN 2.0 A/B controllers
  - Optional full-speed USB 2.0 OTG/ Host/Device
  - Up to 40 PWM outputs
  - Serial communication with up to: 8 UARTs, 6 I2Cs, 4 SPI/SSI
  - Intelligent low-power design power consumption as low as 1.6 µA

4.2 ARM Cortex-M4 – Development Kit

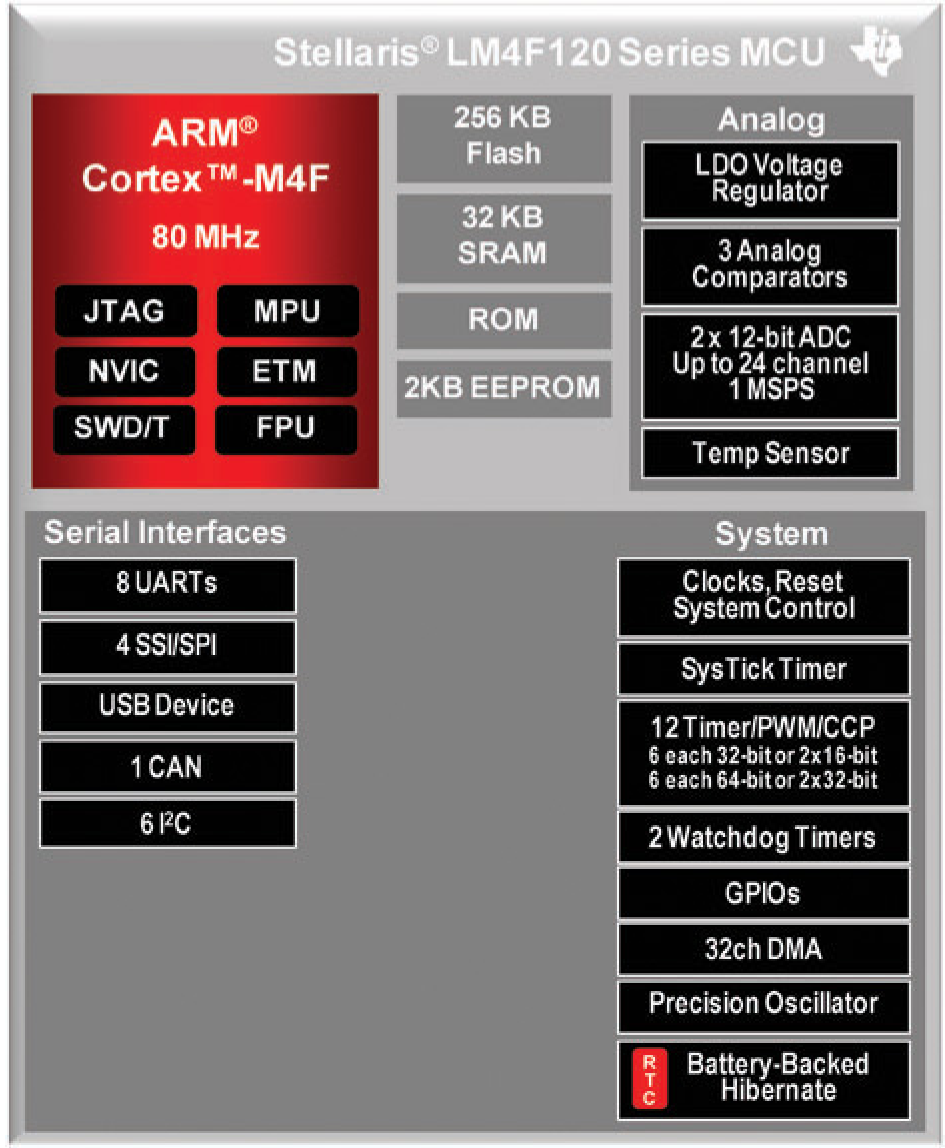
- EK-TM4F120 LaunchPad Evaluation Kit
  - 80-MHz, 32-bit ARM Cortex-M4 CPU
  - 256 Kbytes of FLASH
  - Many peripherals such as MC PWMs, 1-MSPS ADCs, eight UARTs, four SPIs, four I2Cs, USB Host|Device, and up to 27 timers.

- EK-LM4F232 Development Kit
  - ARM® Cortex™-LX4F232
  - color OLED display,
  - USB OTG,
  - A micro SD card, a coin cell battery,
  - A temperature sensor,
  - A three axis



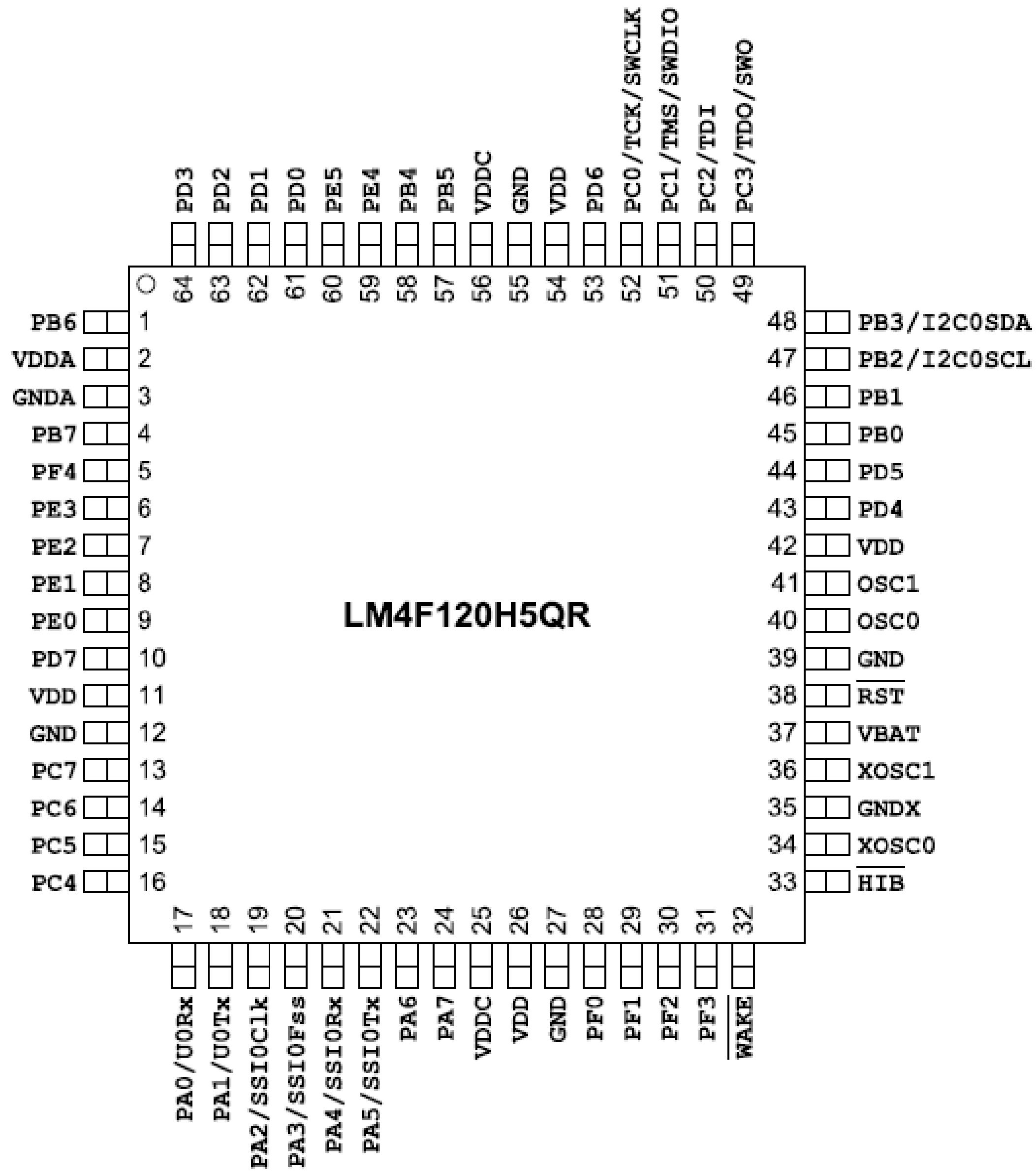
4.2 ARM Cortex-M3 – TM4F

- **Connectivity features:**
  - CAN, USB Device, SPI/SSI, I2C, UARTs
- **High-performance analog integration**
  - Two 1 MSPS 12-bit ADCs
  - Analog and digital comparators
- **Best-in-class power consumption**
  - As low as 370 µA/MHz
  - 500µs wakeup from low-power modes
  - RTC currents as low as 1.7µA
- **Solid roadmap**
  - Higher speeds, Ultra-low power
  - Larger memory



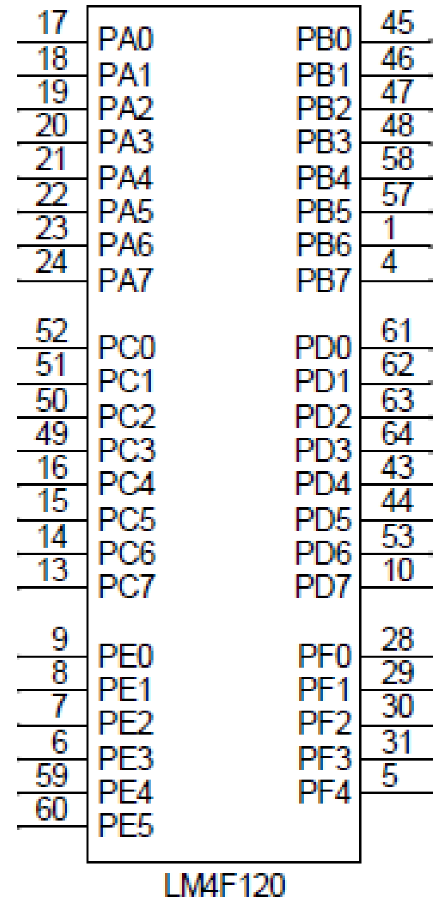
4.3. ARM Cortex-M4 - LM4F120H5QR

- LM4F120H5QR package

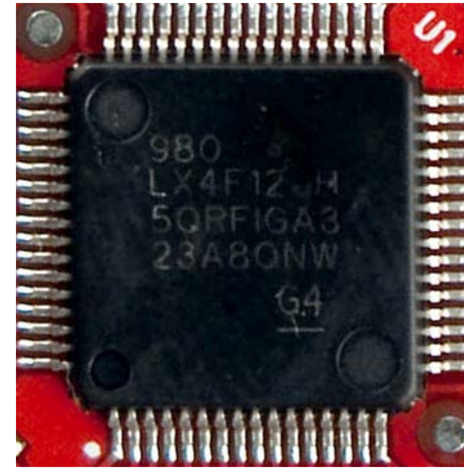


4.3. ARM Cortex-M4 - LM4F120H5QR

- LM4F120H5QR has **6 GPIO blocks**, supporting up to **43 IO pins**
  - Port A: 8 bits
  - Port B : 8 bits
  - Port C : 8 bits
  - Port D : 8 bits
  - Port E : 6 bits
  - Port F : 5 bits
- GPIO pad configuration
  - Weak pull-up or pull-down resistors
  - 2-mA, 4-mA, and 8-mA pad drive
  - Slew rate control for 8-mA pad drive
  - Open drain enables
  - Digital input enables



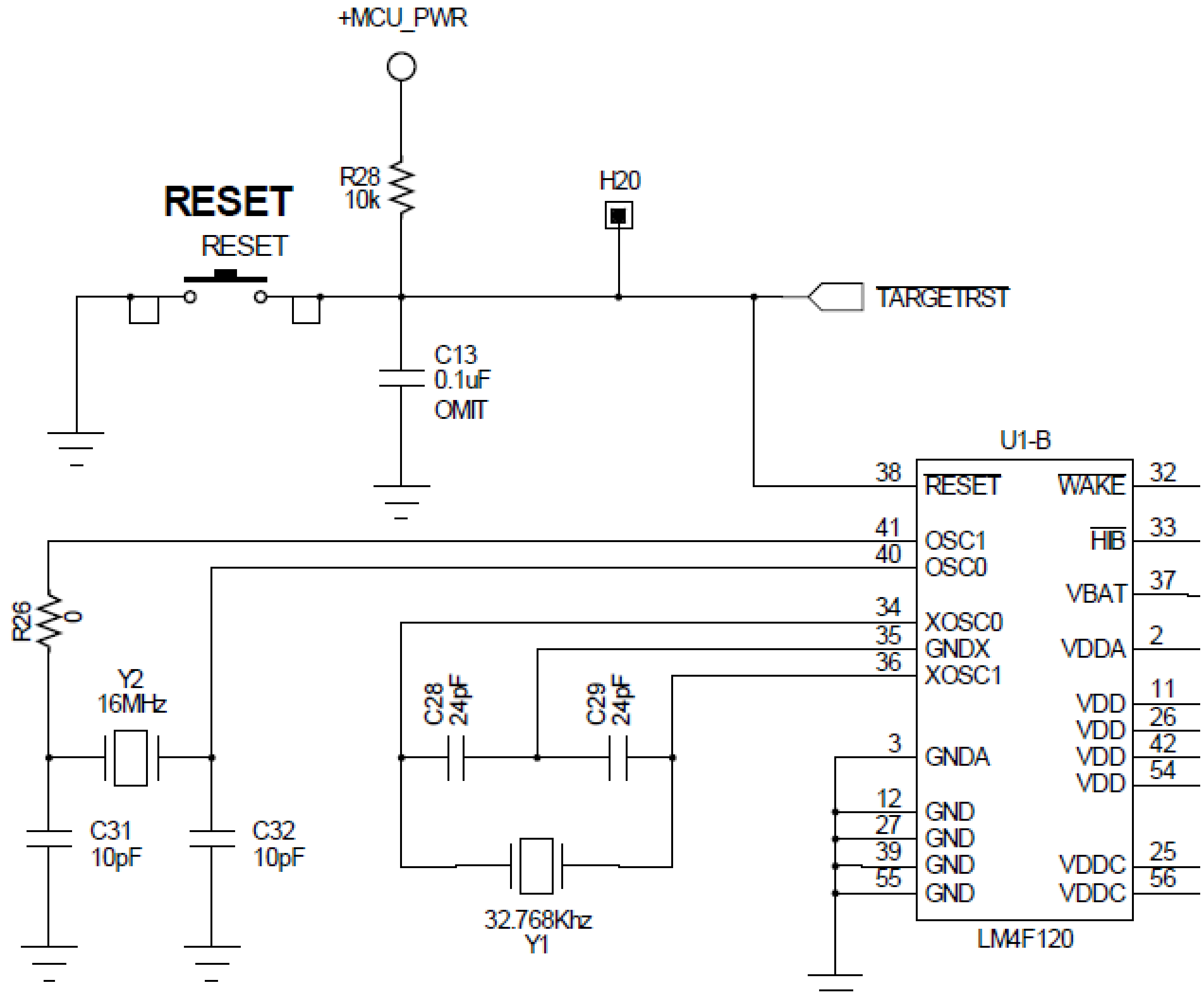
4.3. ARM Cortex-M4 - LM4F120H5QR



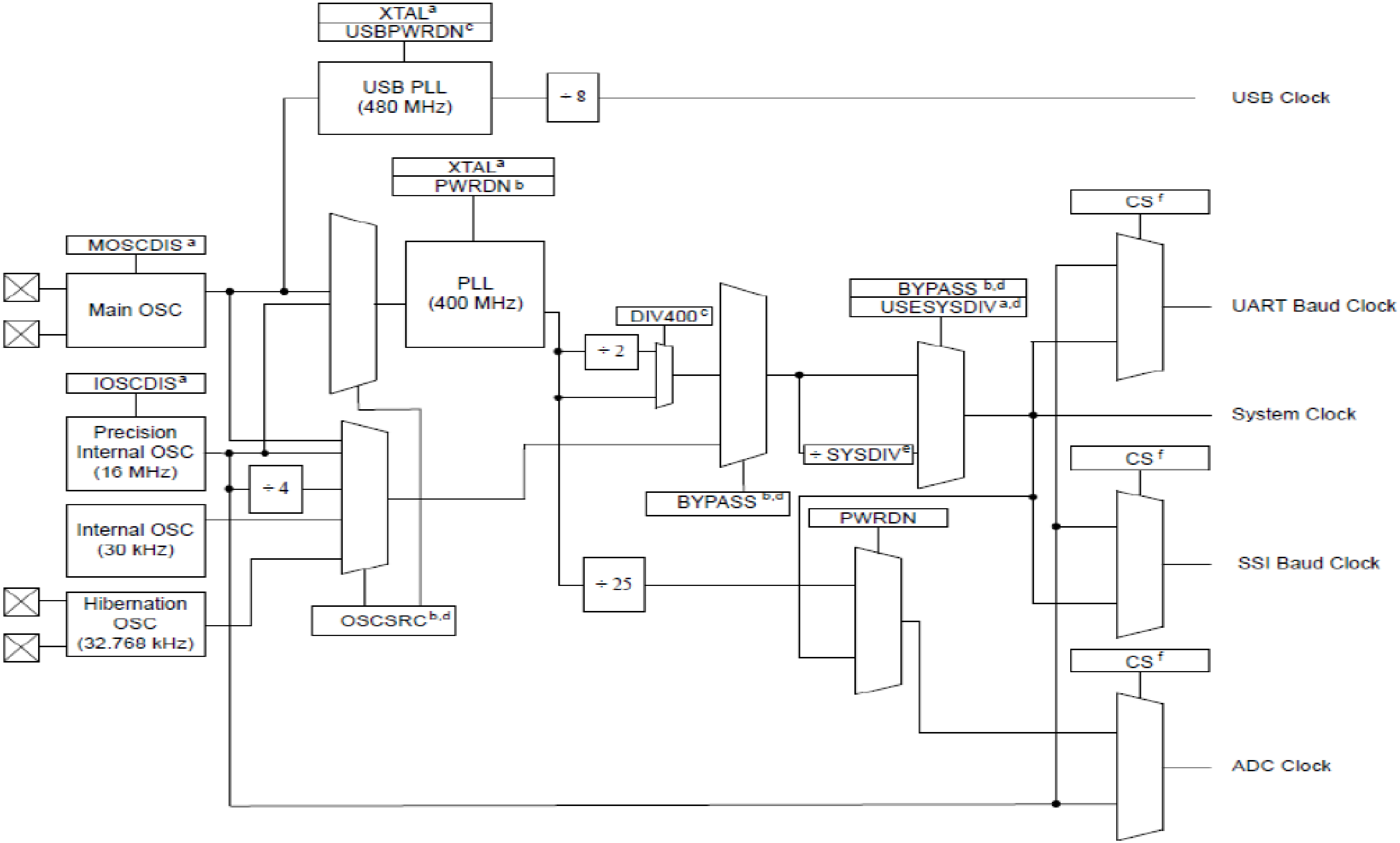
- 256KB Flash memory
  - Single-cycle to **40MHz**
  - Pre-fetch buffer and speculative branch improves performance above 40 MHz
- 32KB single-cycle SRAM with bit-banding
  - Internal ROM loaded with StellarisWare software
  - Stellaris Peripheral Driver Library
  - Stellaris Boot Loader
  - Advanced Encryption Standard (AES) cryptography tables
  - Cyclic Redundancy Check (CRC) error detection functionality
- 2KB EEPROM (fast, saves board space)
  - Wear-leveled 500K program/erase cycles
  - 10 year data retention
  - 4 clock cycle read time

4.3. ARM Cortex-M4 - LM4F120H5QR

- Clock and reset



4.3. ARM Cortex-M4 - LM4F120H5QR



4.3. ARM Cortex-M4 - LM4F120H5QR

USB Device Signals

GPIO Pin	Pin Function	USB Device
PD4	USB0DM	D-
PD5	USB0DP	D+

Stellaris® In-Circuit Debug Interface (ICDI) Signals

GPIO	Pin Pin Function
PC0	TCK/SWCLK
PC1	PC1 TMS/SWDIO
PC2	TDI
PC3	PC3 TDO/SWO

Virtual COM Port Signals

GPIO Pin	Pin Function
PA0	U0RX
PA1	U0TX

4.3. ARM Cortex-M4 - LM4F120H5QR

Virtual COM Port Signals

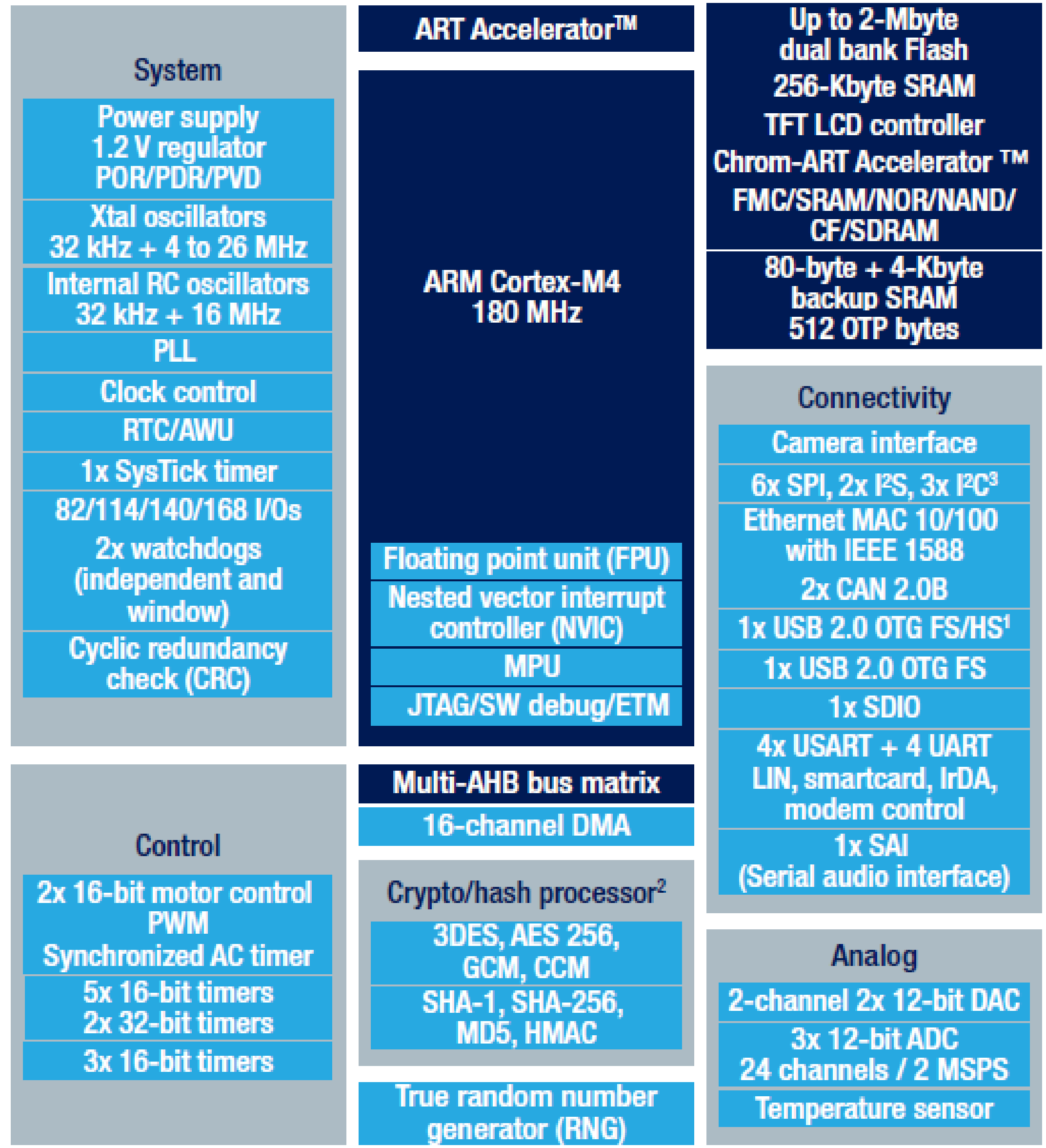
GPIO Pin	Pin Function
PA0	U0RX
PA1	U0TX

4.3. ARM Cortex-M4 – STM32F4

- Features:
  - 180 MHz/225 DMIPS Cortex-M4
  - Single cycle DSP MAC and floating point unit
  - Memory accelerator
  - Graphic accelerator
  - Multi DMA controllers
  - SDRAM interface support
  - Ultra-low dynamic power in Run mode: 260 µA/MHz at 180 MHz

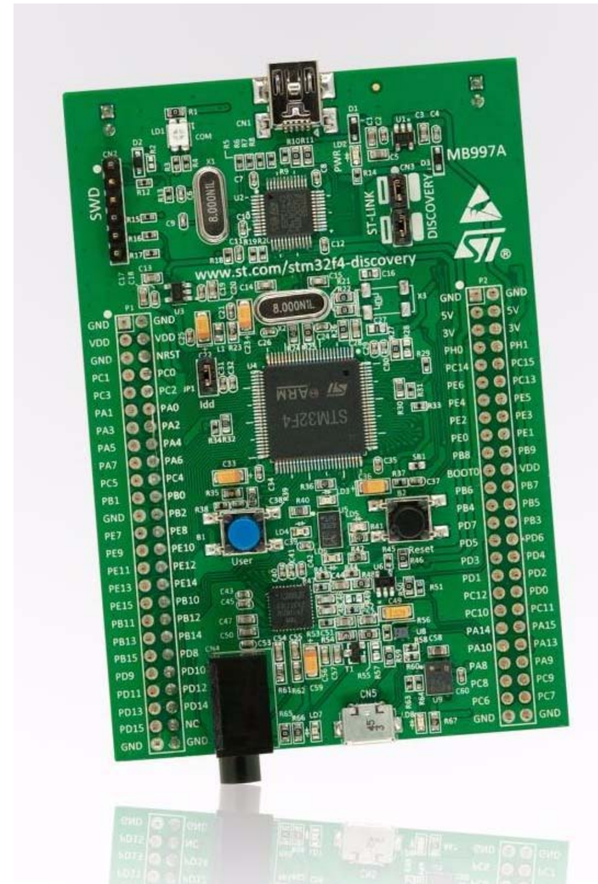
4.3. ARM Cortex-M4 – STM32F4

- Block diagram



4.3. ARM Cortex-M4 – STM32F4

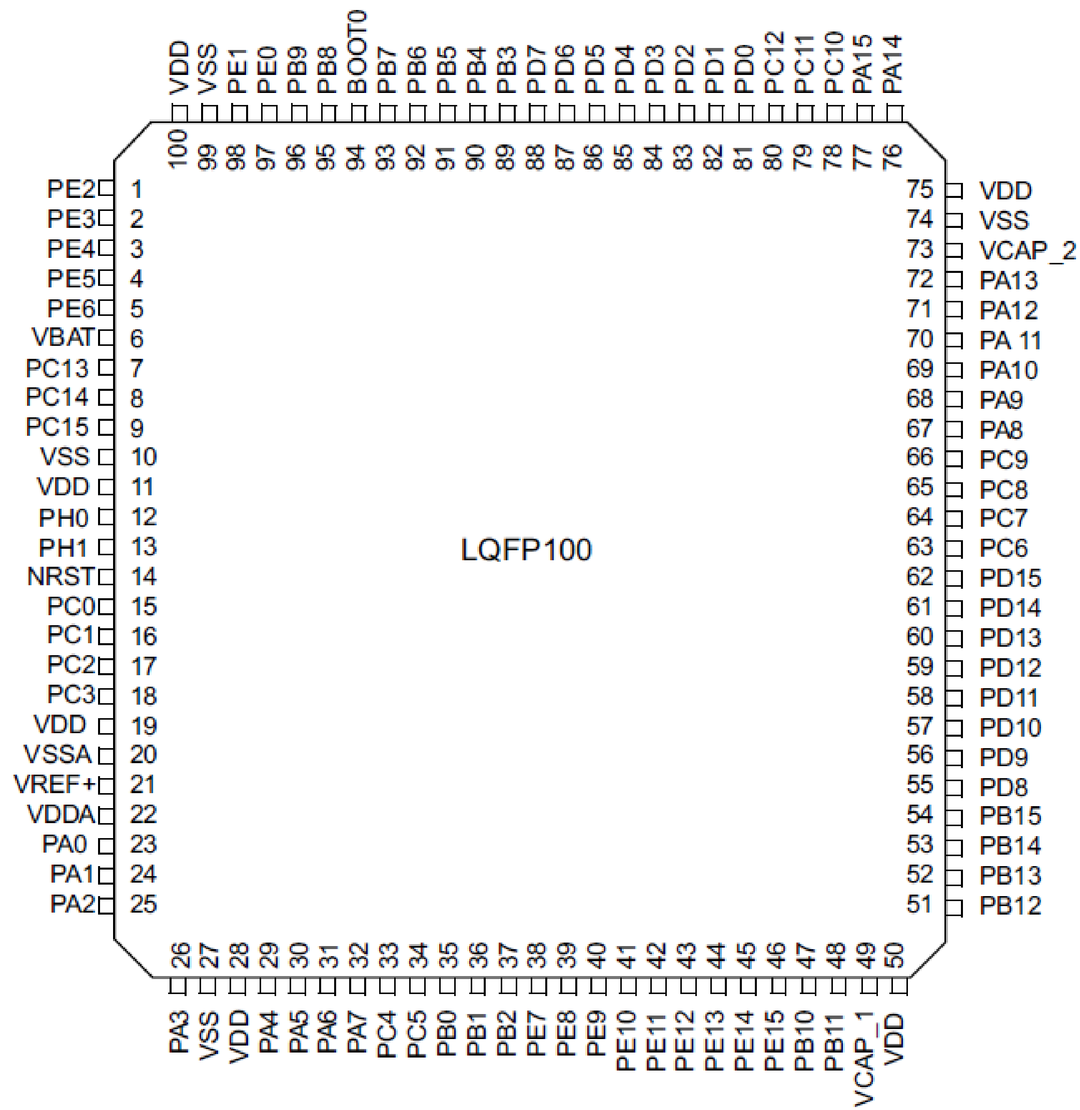
- Development kit
  - STM32F407VGT6 microcontroller
  - 168MHz/210 DMIPS
  - DSP MAC and floating point unit
  - 1 MB Flash, 192 KB RAM
  - On-board ST-LINK/V2
  - Power supply: 3 V and 5 V
  - 3-axis accelerometer
  - Audio sensor, omni-directional digital microphone
  - audio DAC with integrated class D speaker driver
  - Eight LEDs:
  - Two push buttons (user and reset)



Discovery kit for STM32F407

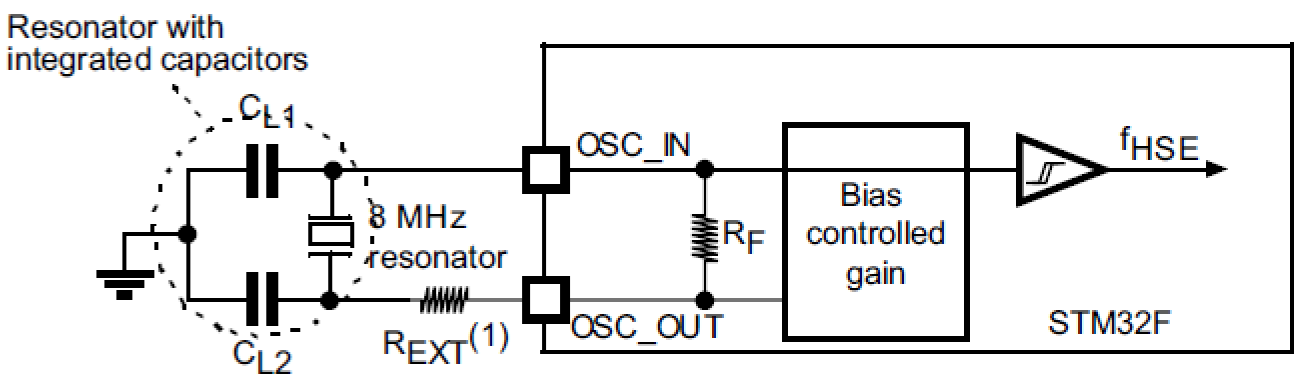
4.3. ARM Cortex-M4 – STM32F4

- Package LQFP100
- GPIO
  - Port A: 16 bit
  - Port B: 16 bit
  - Port C: 16 bit
  - Port D: 16 bit
  - Port E: 16 bit
- can sink or source up to ±8mA
- except PC13, PC14 and PC15 which can sink or source up to ±3mA

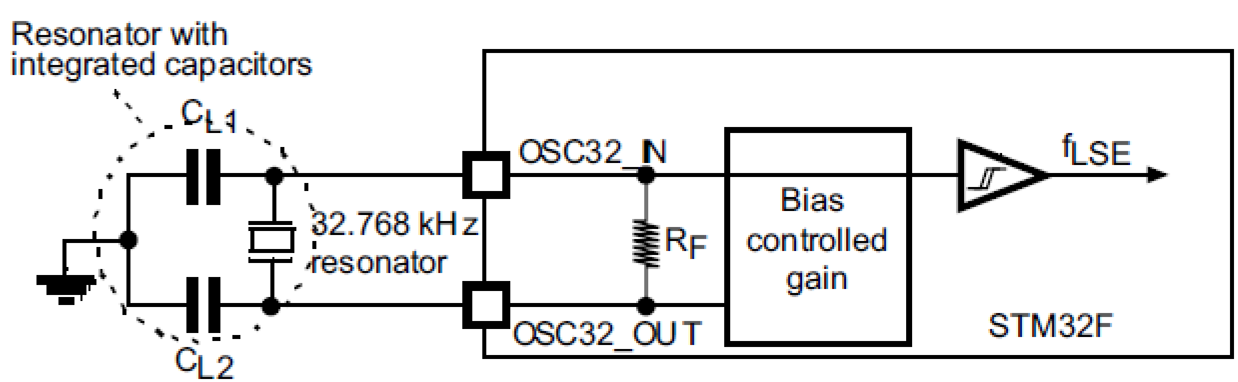


4.3. ARM Cortex-M4 – STM32F4

Typical application with an 8 MHz crystal

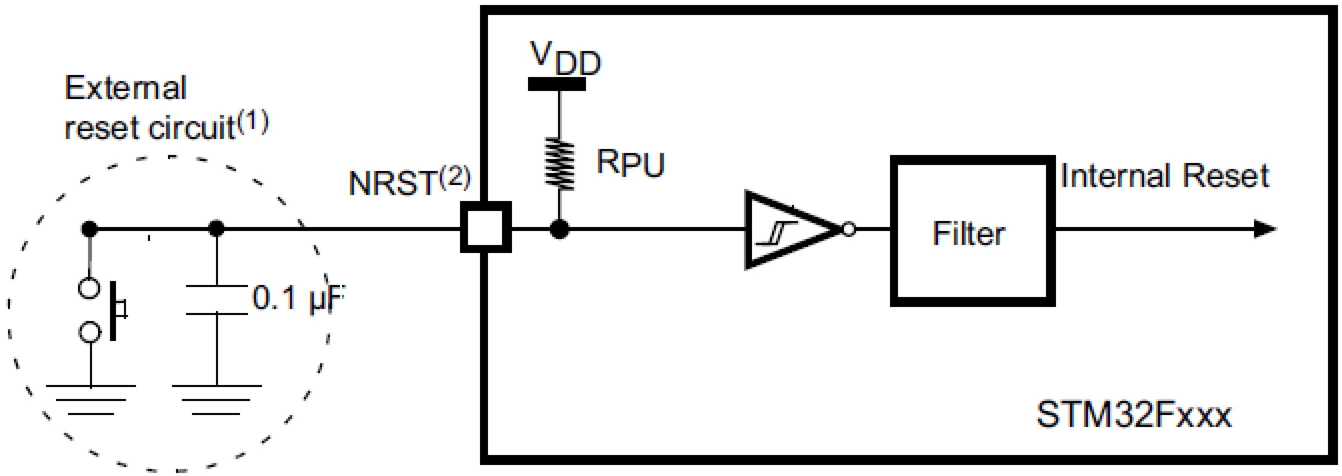


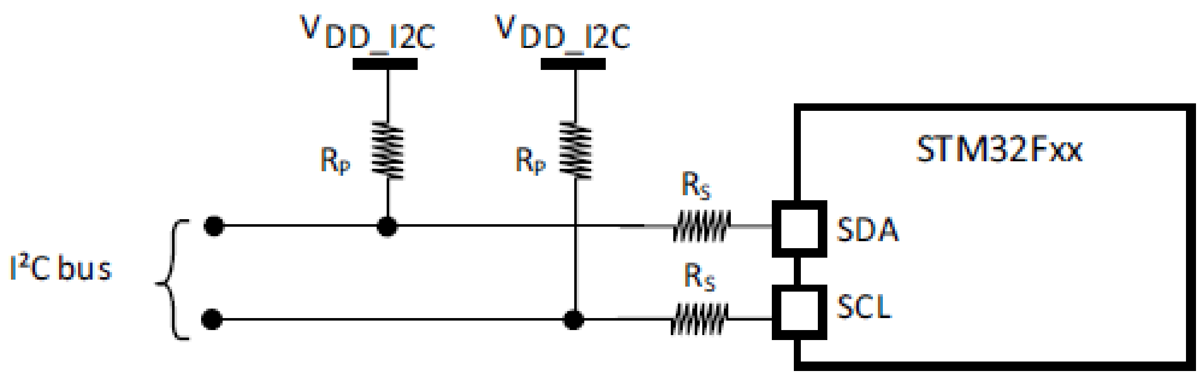
Typical application with a 32.768 kHz crystal



4.3. ARM Cortex-M4 – STM32F4

- Reset circuit



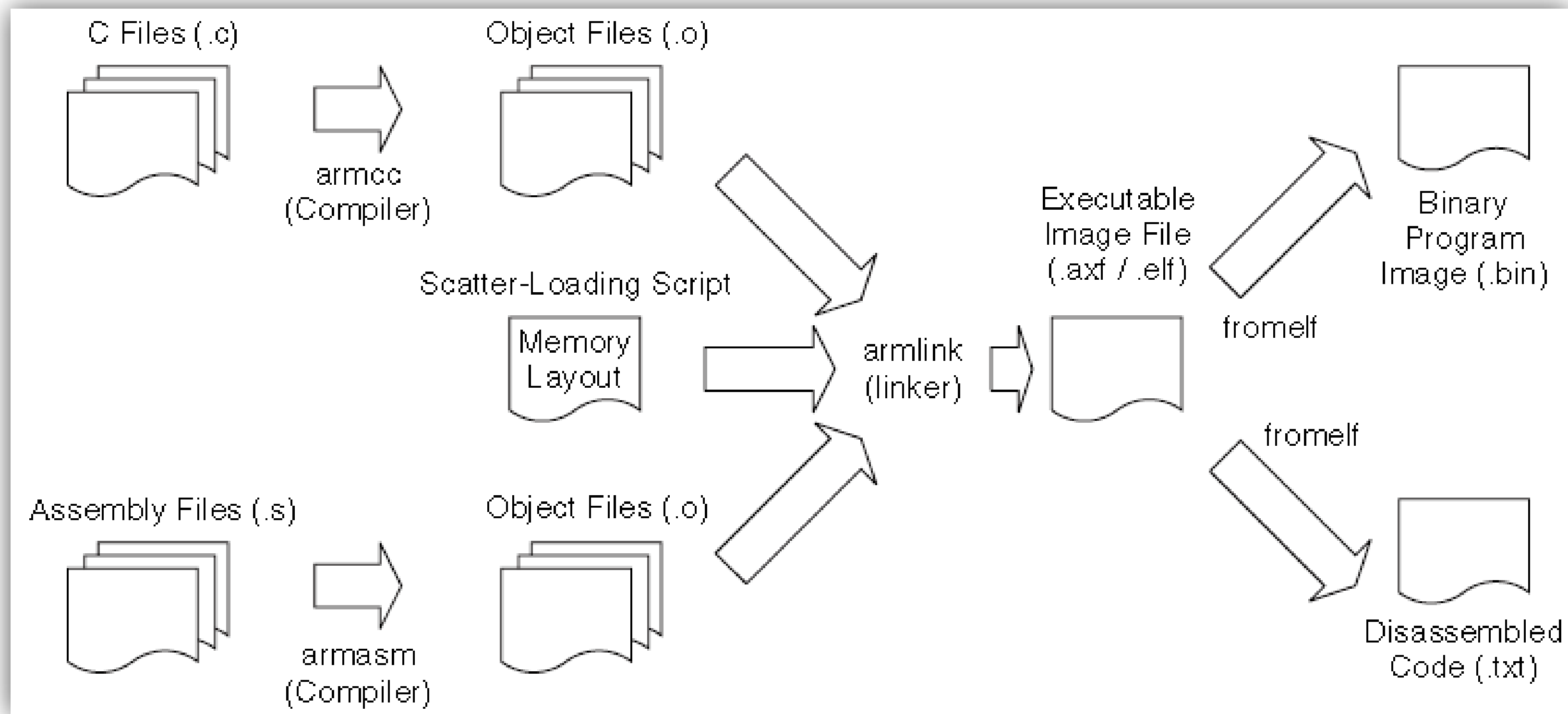


5. ARM Programming

- Using Assembly
  - for small projects
  - can get the best optimization, smallest memory size
  - increase development time, easy to make mistakes
- Using C
  - easier for implementing complex operations
  - larger memory size
  - able to include assembly code (*inline assembler*)
  - Tools: RealView Development Suite (RVDS), KEIL RealView Microcontroller Development Kit, Code Composer, IAR

5. ARM Programming

- Typical development flow



5. ARM Programming – Simple program

This simple program contains the initial SP value, the initial PC value, and setup registers and then does the required calculation in a loop.

```
STACK_TOP      EQU 0x20002000 ; constant for SP starting value
               AREA |Header Code|, CODE
               DCD STACK_TOP    ; Stack top
               DCD Start        ; Reset vector
               ENTRY            ; Indicate program execution start here
Start                           ; Start of main program initialize registers
               MOV r0, #10      ; Starting loop counter value
               MOV r1, #0       ; starting result
loop                            ; Calculated 10+9+8+...+1
               ADD r1, r0       ; R1 = R1 + R0
               SUBS r0, #1      ; Decrement R0, update fl ag ("S" suffi x)
               BNE loop         ; If result not zero jump to loop,
                                ; result is now in R1
deadloop       B deadloop       ; Infinite loop
               END               ; End of file
```

5. ARM Programming - Simple program

- Compile a assemply code
  - armasm --cpu cortex-m3 -o test1.o test1.s
- Link to executable image
  - armlink --rw_base 0x20000000 --ro_base 0x0 --map -o test1.elf test1.o
- Create the binary image
  - fromelf --bin --output test1.bin test1.elf
- generate a disassembled code list file
  - fromelf -c --output test1.list test1.elf

5. ARM Programming

- Using EQU to define constants

```
NVIC_IRQ_SETEN0            EQU     0xE000E100
NVIC_IRQ0_ENABLE           EQU     0x1
          LDR R0,NVIC_IRQ_SETEN0
          MOV R1,#NVIC_IRQ0_ENABLE ; Move immediate data to register
          STR R1, [R0]             ; Enable IRQ 0 by writing R1 to address in R0
```

- Using DCI to code an instruction

```
DCI 0xBE00 ; Breakpoint (BKPT 0), a 16-bit instruction
```

- Using DCB and DCD to define binary data

```
MY_NUMBER
          DCD 0x12345678
HELLO_TXT
          DCB “Hello\n”,0 ; null terminated string
```

5. ARM Programming – Moving data

- Data transfers can be of one of the following types:
  - Moving data between register and register
  - Moving data between memory and register
  - Moving data between special register and register
  - Moving an immediate data value into a register

```
        MOV     R8, R3          ; moving data from register R3 to register R8
        MOV     R0, #0x12       ; Set R0 = 0x12 (hexadecimal)
        MOV     R1, #'A'        ; Set R1 = ASCII character A
```

```
        MRS     R0, PSR         ; Read Processor status word into R0
        MSR     CONTROL, R1     ; Write value of R1 into control register
```

```
        LDR R0, address1        ; R0 set to 0x4001
        ...
address1
        0x4000: MOV R0, R1      ; address1 contains program code
```

5. ARM Programming – Using Stack

- Stack PUSH and POP

```
subroutine_1
            PUSH {R0-R7, R12, R14}      ; Save registers
            … ; Do your processing
            POP {R0-R7, R12, R14}       ; Restore registers
            BX R14                      ; Return to calling function
```

- Link register (LR or R14)

```
main                        ; Main program
            BL function1    ; Call function1 using Branch with Link
                            ; instruction.
                            ; PC  function1 and
                            ; LR  the next instruction in main
            …
function1
            …               ; Program code for function 1
            BX LR ; Return
```

5. ARM Programming – Special Register

- Special registers can only be accessed via MSR and MRS instructions

```
MRS     <reg>, <special_reg>        ; Read special register
MSR     <special_reg>, <reg>        ; write to special register
```

- ASP can be changed by using MSR instruction, but EPSR and IPSR are read-only

```
MRS     r0, APSR      ; Read Flag state into R0
MRS     r0, IPSR      ; Read Exception/Interrupt state
MRS     r0, EPSR      ; Read Execution state
MSR     APSR, r0      ; Write Flag state
MRS     r0, PSR       ; Read the combined program status word
MSR     PSR, r0       ; Write combined program state word
```

5. ARM Programming – Special Register

- To access the Control register, the MRS and MSR instructions are used:

```
MRS    r0, CONTROL     ; Read CONTROL register into R0
MSR    CONTROL, r0     ; Write R0 into CONTROL register
```

Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.
CONTROL[0]	0 = Privileged in Thread mode 1 = User state in Thread mode If in handler mode (not Thread mode), the processor operates in privileged mode.

5. ARM Programming

- 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

5. ARM Programming

- 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)

5. ARM Programming – 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

5. ARM Programming – IF-THEN

- The IF-THEN (IT) instructions allow up to four succeeding instructions (called an *IT block) to* be conditionally executed.
- They are in the following formats:

IT<x> <cond>
IT<x><y> <cond>
IT<x><y><z> <cond>
where:
- *<x > specifies the execution condition for the second instruction*
- *<y > specifies the execution condition for the third instruction*
- *<z > specifies the execution condition for the fourth instruction*

5. ARM Programming – IF-THEN

Symbol	Condition	Flag
EQ	Equal	Z set
NE	Not equal	Z clear
CS/HS	Carry set/unsigned higher or same	C set
CC/LO	Carry clear/unsigned lower	C clear
MI	Minus/negative	N set
PL	Plus/positive or zero	N clear
VS	Overflow	V set
VC	No overflow	V clear
HI	Unsigned higher	C set and Z clear
LS	Unsigned lower or same	C clear or Z set
GE	Signed greater than or equal	N set or V set, or N clear and V clear (N == V)
LT	Signed less than	N set and V clear, or N clear and V set (N != V)
GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0, N == V)
LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)
AL	Always (unconditional)	—

5. ARM Programming – IF-THEN

- An example of a simple conditional execution

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

- In assembly:

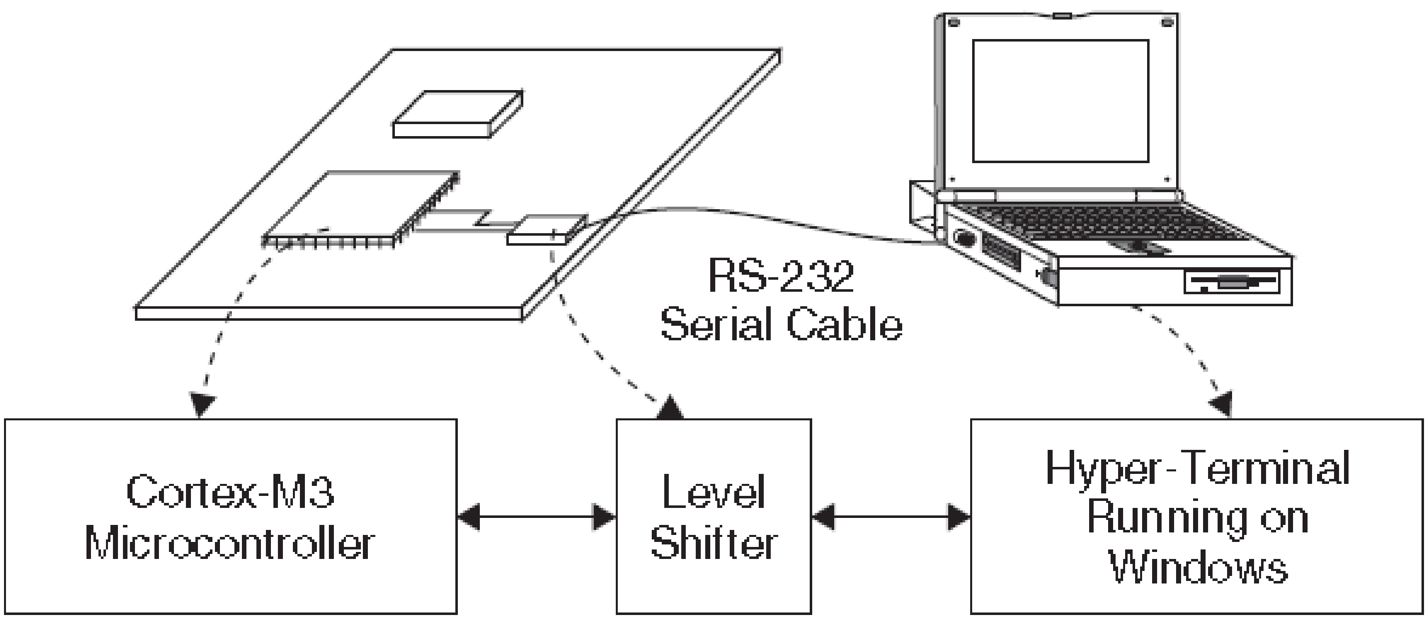
```
CMP      R1, R2    ; If R1 < R2 (less then)
ITTEE    LT        ; then execute instruction 1 and 2
                   ; (indicated by T)
                   ; else execute instruction 3 and 4
                   ; (indicated by E)
SUBLT.W  R2,R1     ; 1st instruction
LSRLT.W  R2,#1     ; 2nd instruction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

5. ARM Programming – Using Data Memory

```
STACK_TOP    EQU 0x20002000    ; constant for SP starting value
             AREA |Header Code|, CODE
             DCD STACK_TOP     ; SP initial value
             DCD Start         ; Reset vector
             ENTRY
Start        ; Start of main program, initialize registers
             MOV r0, #10       ; Starting loop counter value
             MOV r1, #0        ; starting result. Calculated 10+9+8+...+1
loop         ADD r1, r0        ; R1 = R1 + R0
             SUBS r0, #1       ; Decrement R0, update fl ag ("S" suffi x)
             BNE loop          ; If result not zero jump to loop; Result is now in R1
             LDR r0,=MyData1   ; Put address of MyData1 into R0
             STR r1,[r0]       ; Store the result in MyData1
deadloop     B deadloop        ; Infi nite loop
             AREA |Header Data|, DATA
             ALIGN 4
MyData1      DCD 0             ; Destination of calculation result
MyData2      DCD 0
             END               ; End of file
```

5. ARM Programming

- A Low-Cost Test Environment for Outputting Text Messages
  - UART interface is common output method to send messages to a console
  - Hyper-Terminal program can be used as a console



5. ARM Programming

- A simple routine to output a character through UART

```
UART0_BASE        EQU 0x4000C000
UART0_FLAG        EQU UART0_BASE+0x018
UART0_DATA        UART0_BASE+0x000
Putc                              ; Subroutine to send a character via UART
                                  ; Input R0 = character to send
        PUSH {R1,R2, LR}          ; Save registers
        LDR R1,=UART0_FLAG
PutcWaitLoop
        LDR R2,[R1]               ; Get status flag
        TST R2, #0x20             ; Check transmit buffer full flag bit
        BNE PutcWaitLoop ; If busy then loop
        LDR R1,=UART0_DATA ; otherwise
        STRB R0, [R1]             ; Output data to transmit buffer
        POP {R1,R2, PC}           ; Return
```

The register addresses and bit definitions here are just examples

TI's ARM Cortex-M Development Kit

- LM3S9B96 development Kit
  - Stellaris LM3S9B96 MCU with fully-integrated Ethernet, CAN, and USB OTG/Host/Device
  - Bright 3.5" QVGA LCD touch-screen display
  - Navigation POT switch and select pushbuttons
  - Integrated Interchip Sound (I2S) Audio Interface

- The Tiva C Series EK-TM4C123GXL LaunchPad Evaluation Kit
  - A TM4C123G LaunchPad Evaluation board
  - On-board In-Circuit Debug Interface (ICDI)
  - USB Micro-B plug to USB-A plug cable
  - Preloaded RGB quickstart application
  - ReadMe First quick-start guide

Quiz

1. What are different features between ARM Cortex M3 and M4?
2. What are differences between Thumb and Thumb-2 instructions?
3. Compare the features between TM4C and STM32F4 microcontroller
4. What are extra instructions that ARM Cortex-M4 supports?

Assignments

1. Write a program to move 10 words from 0x20000000 to 0x3000000.
2. Write a program to read STATUS register and write to 0x20000004
3. Write a program to write a value in 0x30000000 to CONTROL register
4. Write a subroutine to perform a function 40*X + 50
5. Write a subroutine to convert data of 10 words form big endian to little endian.
6. Write a program as pseudo code below:

```
if (R0 equal R1) then {
        R3 = R4 + R5
        R3 = R3 / 2  }
else {
        R3 = R6 + R7
        R3 = R3 / 2
}
```

Assignments

- Design a circuit described as follows:
  – Using Cortex-M4 processor LM4F120H5QR
  – Port A connects to 8 single LEDs
  – Port B connects to 8 buttons
  – Write a program to control 8 LEDs by 8 buttons

Assignments

- Design a circuit described as follows:
  - Using Cortex-M4 processor STM32F407VGT6
  - Port A connects to a character LCD
  - Port B connects to 3 buttons START, STOP, CLEAR
  - Write a program to control as follows:
    - START: start to count number in millisecond
    - STOP: stop to count
    - CLEAR: clear the number to zero