Real Time Systems – SS2016

Prof. Dr. Karsten Weronek

Faculty 2

Computer Science and Engineering

STR9 microcontroller

Our Microcontroller in the Exercises

In the exercises we will configure microcontroller.

We use the STR9 by STM.

This microcontroller is based on ARM966-E core at 96MHz.

It is old, inexpensive but has a bunch of components.

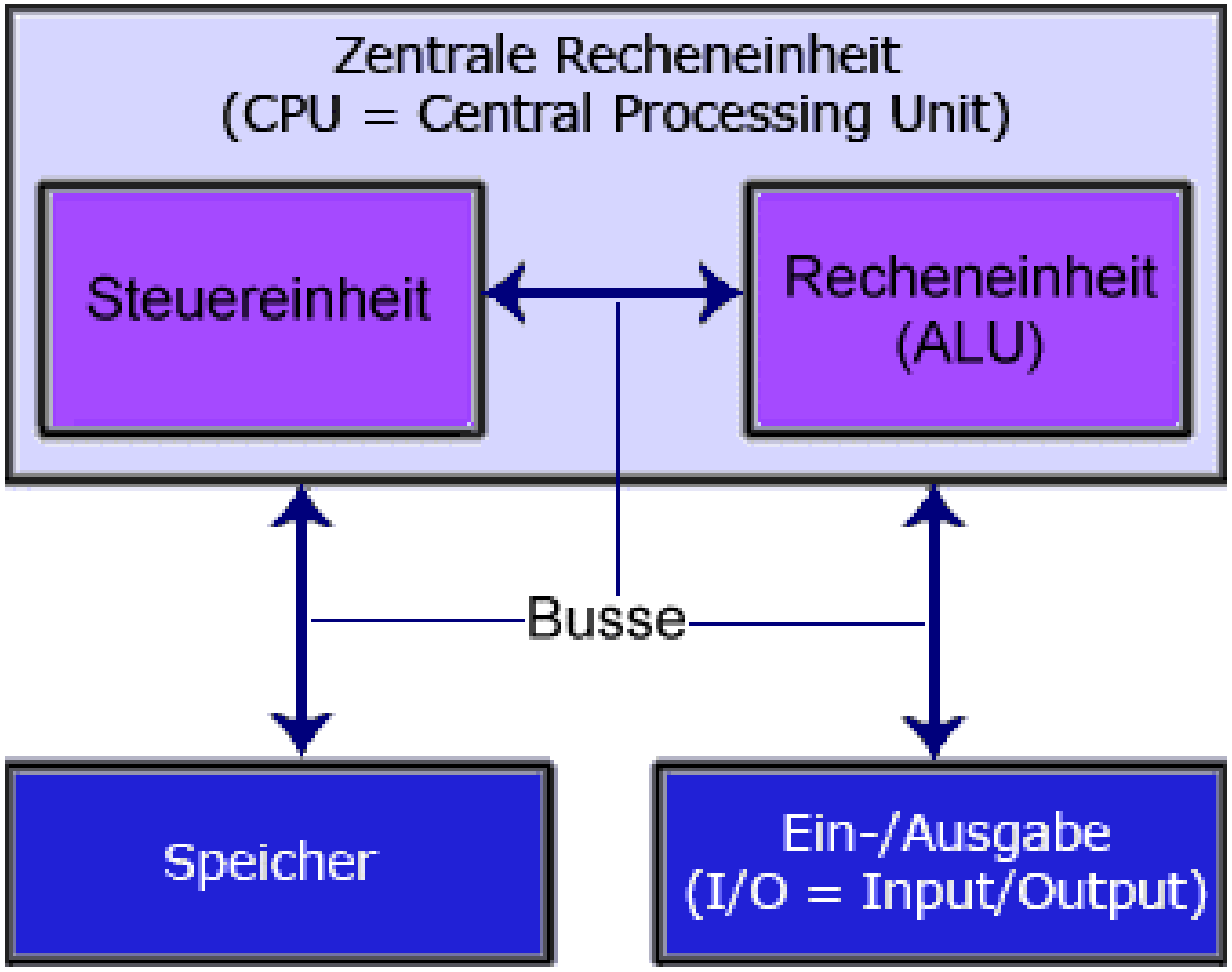
ARM has no production plants, they just sell licenses for their design.

Our chip was produced by ST Microelectronics.

This chip is not based on a "von Neumann Architecture" but on an "Havard Architecture": different Memory for Data and Code

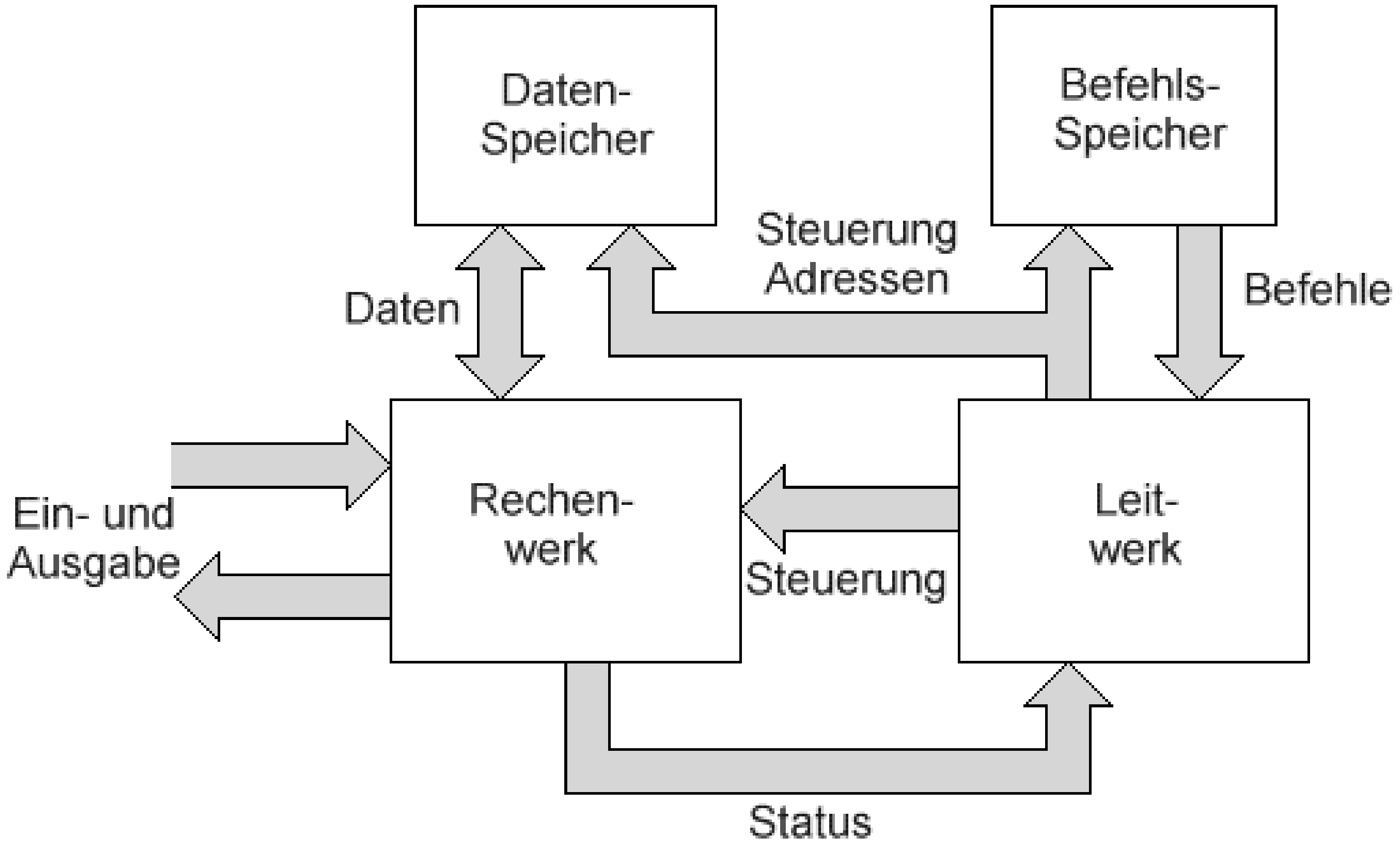
The Havard Architecture is faster!

Von Neumann Architecture



ref.: http://www.netzmafia.de/skripten/mikrocomputer/kap1.html

Harvard Architecture



ref.: http://www.netzmafia.de/skripten/mikrocomputer/kap1.html

Features

16/32-bit 96 MHz ARM9E based MCU

ARM966E-S RISC core:

Harvard architecture

5-stage pipeline

Tightly-Coupled Memories (SRAM and Flash)

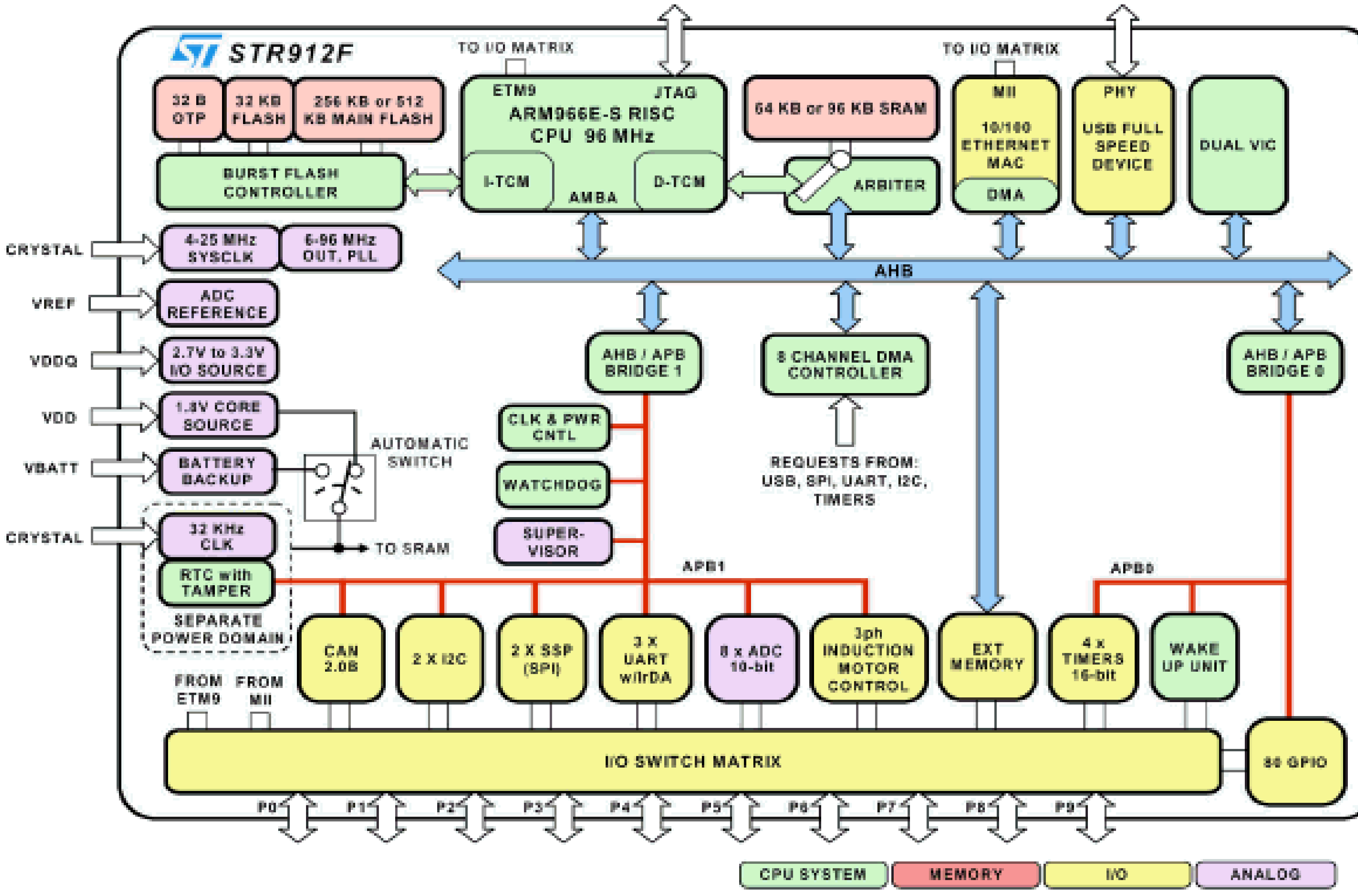
Up to 96 MIPS directly from Flash memory

Single-cycle DSP instructions supported

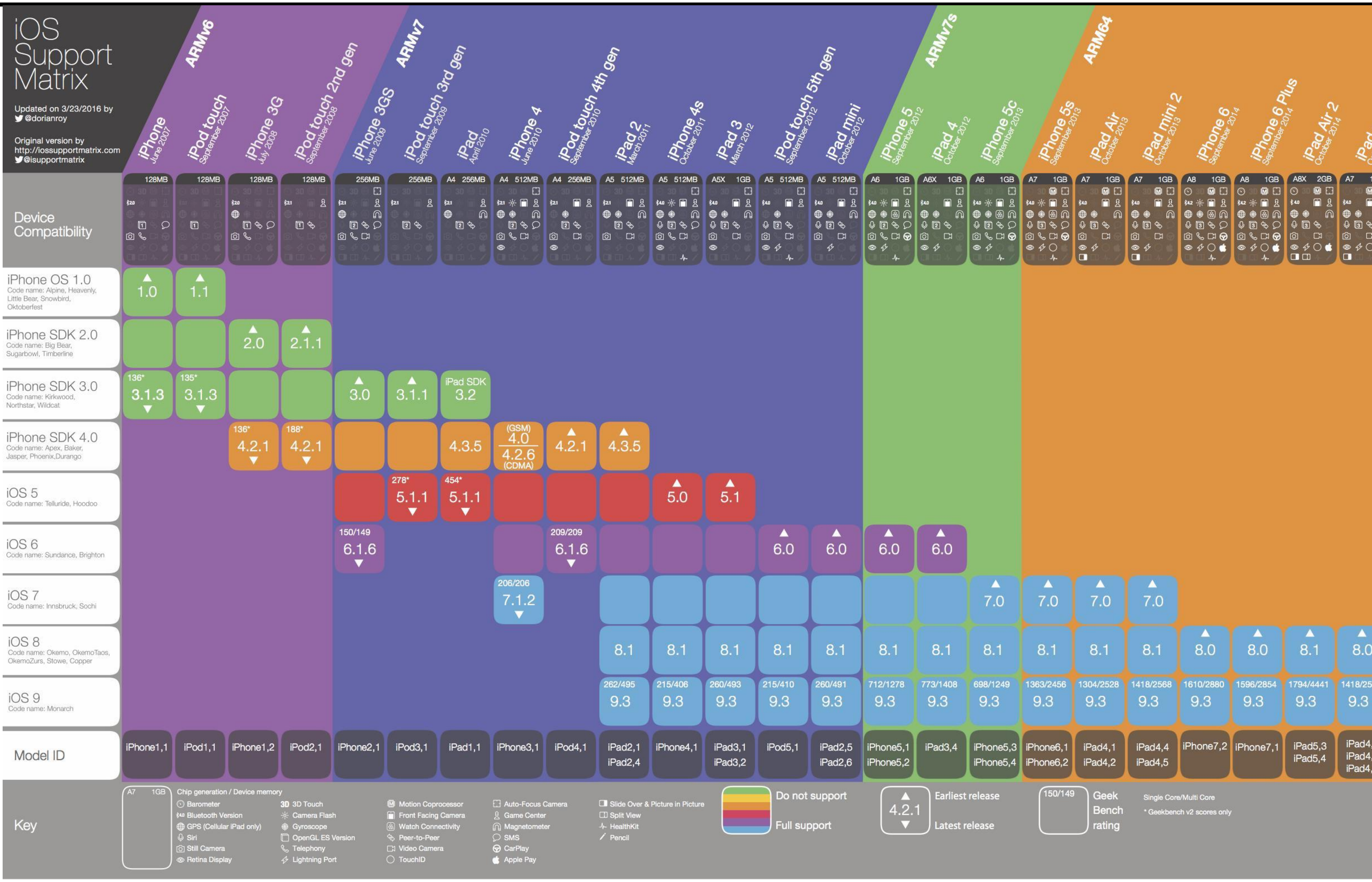
Binary compatible with ARM7 code

Dual burst Flash memories, 32-bits wide

Achitecture



Just to see ....



iOS Support Matrix

Updated on 3/23/2016 by @dorianroy

Original version by http://iossupportmatrix.com @isupportmatrix

Device	Architecture	Device Compatibility	iPhone OS 1.0 (Code name: Alpine, Heavenly, Little Bear, Snowbird, Oktoberfest)	iPhone SDK 2.0 (Code name: Big Bear, Sugarbowl, Timberline)	iPhone SDK 3.0 (Code name: Kirkwood, Northstar, Wildcat)	iPhone SDK 4.0 (Code name: Apex, Baker, Jasper, Phoenix, Durango)	iOS 5 (Code name: Telluride, Hoodoo)	iOS 6 (Code name: Sundance, Brighton)	iOS 7 (Code name: Innsbruck, Sochi)	iOS 8 (Code name: Okemo, OkemoTaos, OkemoZurs, Stowe, Copper)	iOS 9 (Code name: Monarch)	Model ID
iPhone, June 2007	ARMv6	128MB	▲ 1.0	🟩	136* 3.1.3 ▼							iPhone1,1
iPod touch, September 2007	ARMv6	128MB	▲ 1.1	🟩	135* 3.1.3 ▼							iPod1,1
iPhone 3G, July 2008	ARMv6	128MB		▲ 2.0	🟩	136* 4.2.1 ▼						iPhone1,2
iPod touch 2nd gen, September 2008	ARMv6	128MB		▲ 2.1.1	🟩	188* 4.2.1 ▼						iPod2,1
iPhone 3GS, June 2009	ARMv7	256MB			▲ 3.0	🟧	🟥	150/149 6.1.6 ▼				iPhone2,1
iPod touch 3rd gen, September 2009	ARMv7	256MB			▲ 3.1.1	🟧	278* 5.1.1 ▼					iPod3,1
iPad, April 2010	ARMv7	A4 256MB			iPad SDK 3.2	4.3.5	454* 5.1.1 ▼					iPad1,1
iPhone 4, June 2010	ARMv7	A4 512MB				(GSM) 4.0 / 4.2.6 (CDMA)	🟥	🟪	206/206 7.1.2 ▼			iPhone3,1
iPod touch 4th gen, September 2010	ARMv7	A4 256MB				▲ 4.2.1	🟥	209/209 6.1.6 ▼				iPod4,1
iPad 2, March 2011	ARMv7	A5 512MB				▲ 4.3.5	🟥	🟪	🟦	8.1	262/495 9.3	iPad2,1 iPad2,4
iPhone 4s, October 2011	ARMv7	A5 512MB					▲ 5.0	🟪	🟦	8.1	215/406 9.3	iPhone4,1
iPad 3, March 2012	ARMv7	A5X 1GB					▲ 5.1	🟪	🟦	8.1	260/493 9.3	iPad3,1 iPad3,2
iPod touch 5th gen, September 2012	ARMv7	A5 512MB						▲ 6.0	🟦	8.1	215/410 9.3	iPod5,1
iPad mini, October 2012	ARMv7	A5 512MB						▲ 6.0	🟦	8.1	260/491 9.3	iPad2,5 iPad2,6
iPhone 5, September 2012	ARMv7s	A6 1GB						▲ 6.0	🟦	8.1	712/1278 9.3	iPhone5,1 iPhone5,2
iPad 4, October 2012	ARMv7s	A6X 1GB						▲ 6.0	🟦	8.1	773/1408 9.3	iPad3,4
iPhone 5c, September 2013	ARMv7s	A6 1GB							▲ 7.0	8.1	698/1249 9.3	iPhone5,3 iPhone5,4
iPhone 5s, September 2013	ARM64	A7 1GB							▲ 7.0	8.1	1363/2456 9.3	iPhone6,1 iPhone6,2
iPad Air, October 2013	ARM64	A7 1GB							▲ 7.0	8.1	1304/2528 9.3	iPad4,1 iPad4,2
iPad mini 2, October 2013	ARM64	A7 1GB							▲ 7.0	8.1	1418/2568 9.3	iPad4,4 iPad4,5
iPhone 6, September 2014	ARM64	A8 1GB								▲ 8.0	1610/2880 9.3	iPhone7,2
iPhone 6 Plus, September 2014	ARM64	A8 1GB								▲ 8.0	1596/2854 9.3	iPhone7,1
iPad Air 2, October 2014	ARM64	A8X 2GB								▲ 8.1	1794/4441 9.3	iPad5,3 iPad5,4
iPad	ARM64	A7								▲ 8.0	1418/25 9.3	iPad4, iPad4, iPad4,

Key

A7 1GB — Chip generation / Device memory

Barometer	3D Touch	Motion Coprocessor	Auto-Focus Camera
Bluetooth Version	Camera Flash	Front Facing Camera	Game Center
GPS (Cellular iPad only)	Gyroscope	Watch Connectivity	Magnetometer
Siri	OpenGL ES Version	Peer-to-Peer	SMS
Still Camera	Telephony	Video Camera	CarPlay
Retina Display	Lightning Port	TouchID	Apple Pay

Slide Over & Picture in Picture
Split View
HealthKit
Pencil

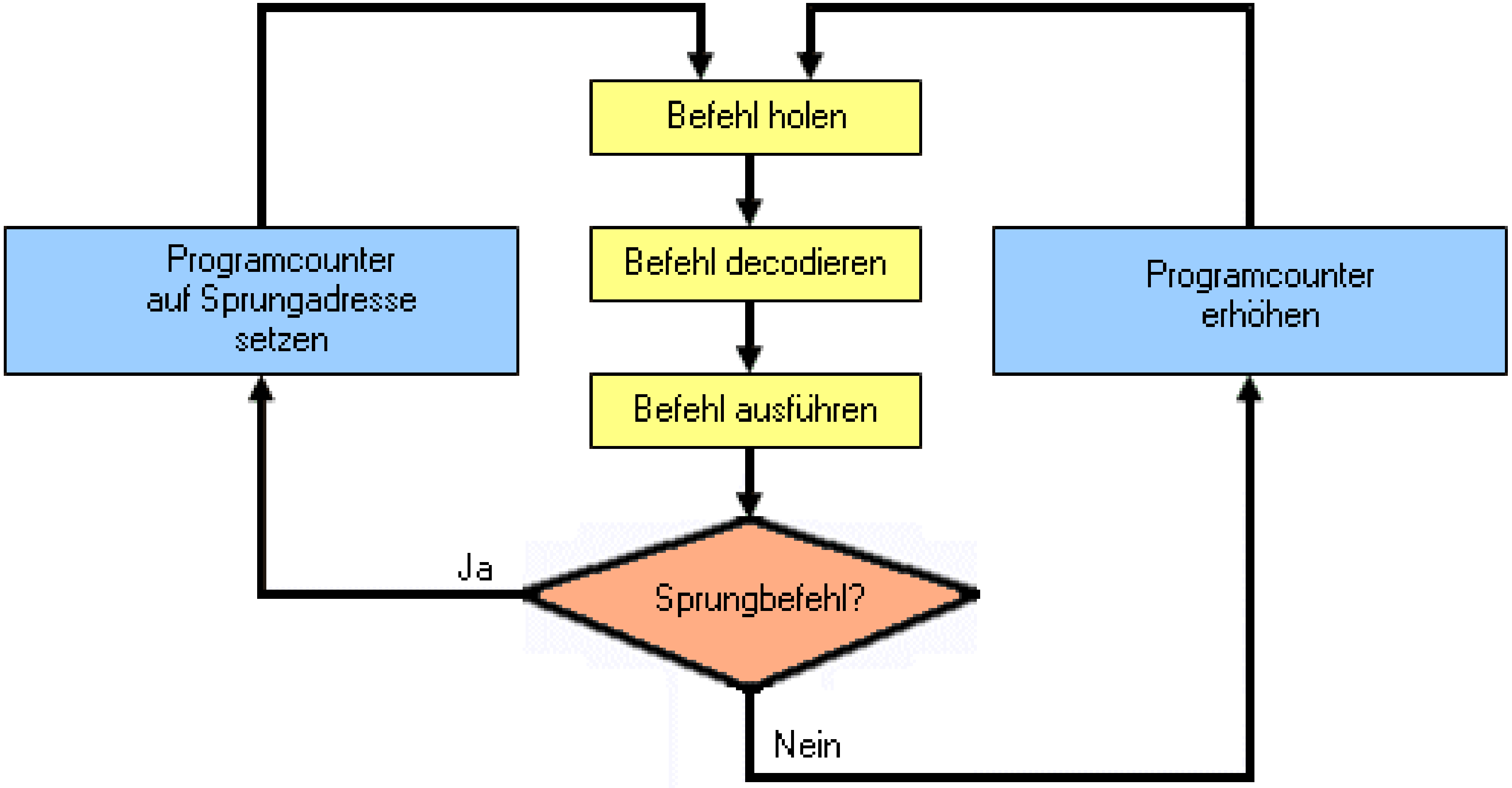
Do not support / Full support

▲ 4.2.1 ▼ — Earliest release / Latest release

150/149 — Geek Bench rating — Single Core/Multi Core

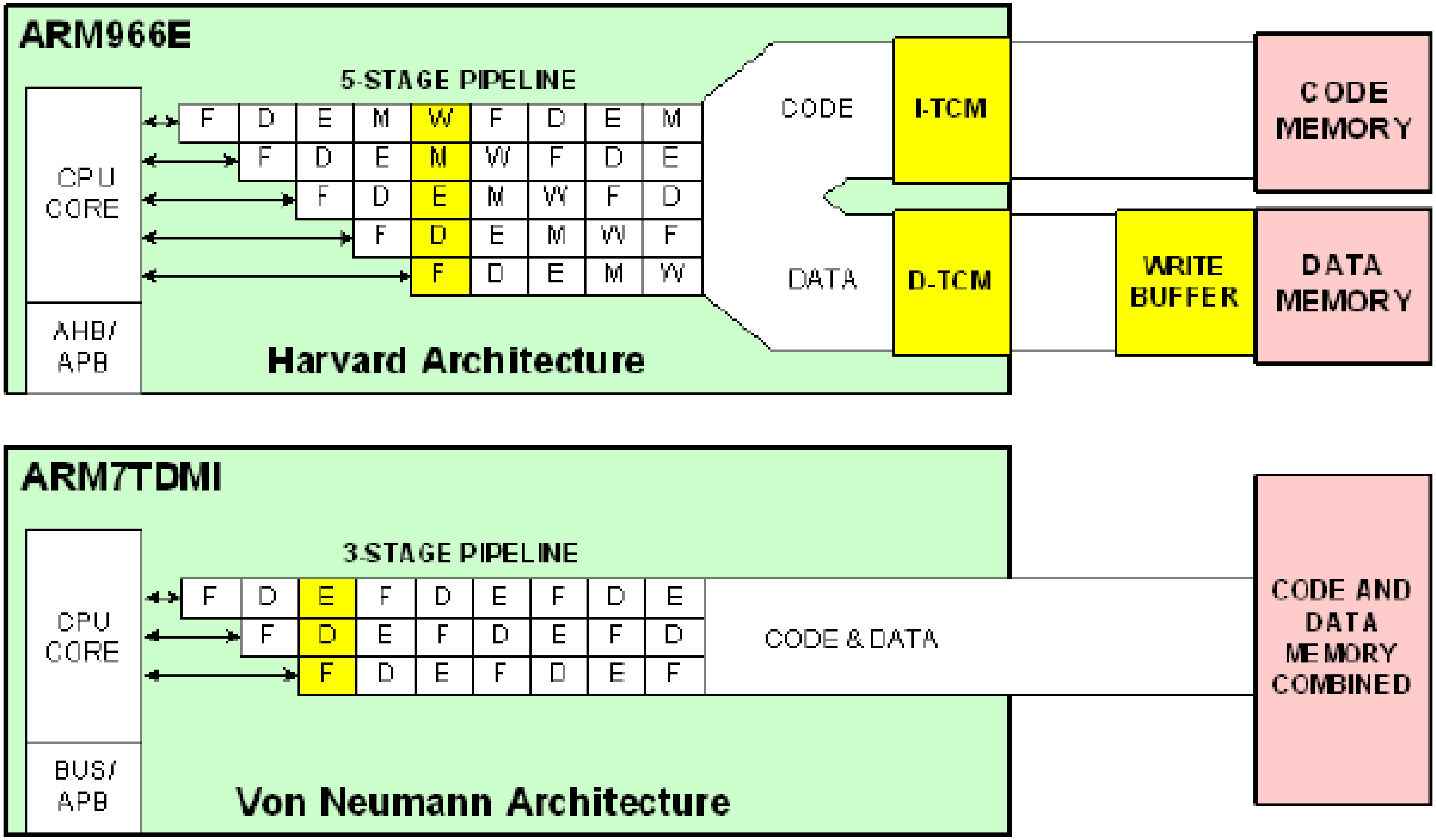
* Geekbench v2 scores only

Processor cycle



The Pipeline

At the heart of the ARM9 CPU is the instruction pipeline. The pipeline is used to process instructions taken from the program store. On the ARM7 a three-stage pipeline is used.



The ARM9 five-stage pipeline has independent fetch, decode, execute memory (read) and (memory) write stages

Arm STR9 - System Control Unit (SCU)

Peripheral clock gating register 0 (SCU_PCGR0)

Address offset: 14h

Reset value: 0000 00DBh

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				MAC	USB 48M	USB	DMA	EXT_MEM _CLK	EMI	VIC	SRAM _ARBITER	SRAM	Res.	PFQ BC	FMI
				rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Peripheral clock gating register 1 (SCU_PCGR1)

Address offset: 18h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							RTC	GPIO 9	GPIO 8	GPIO 7	GPIO 6	GPIO 5	GPIO 4	GPIO 3	GPIO 2
							rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO 1	GPIO 0	WIU	Res.	ADC	CAN	SSP 1	SSP 0	I2C 1	I2C 0	UAR T2	UAR T1	UAR T0	MC	TIM2 3	TIM0 1
rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Arm STR9 - System Control Unit (SCU)

Peripheral reset register 0 (SCU_PRR0)

Address offset: 1Ch

Reset value: 0000 1053h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			RST_ PFQBC_ AHB	RST_ MAC	Res.	RST_ USB	RST_ DMA	Res.	RST_ EMI	RST_ VIC	RST_ SRAM_ ARBITER	Res.		RST_ PFQBC	RST_ FMI
			rw	rw		rw	rw		rw	rw	rw			rw	rw

Peripheral reset register 1 (SCU_PRR1)

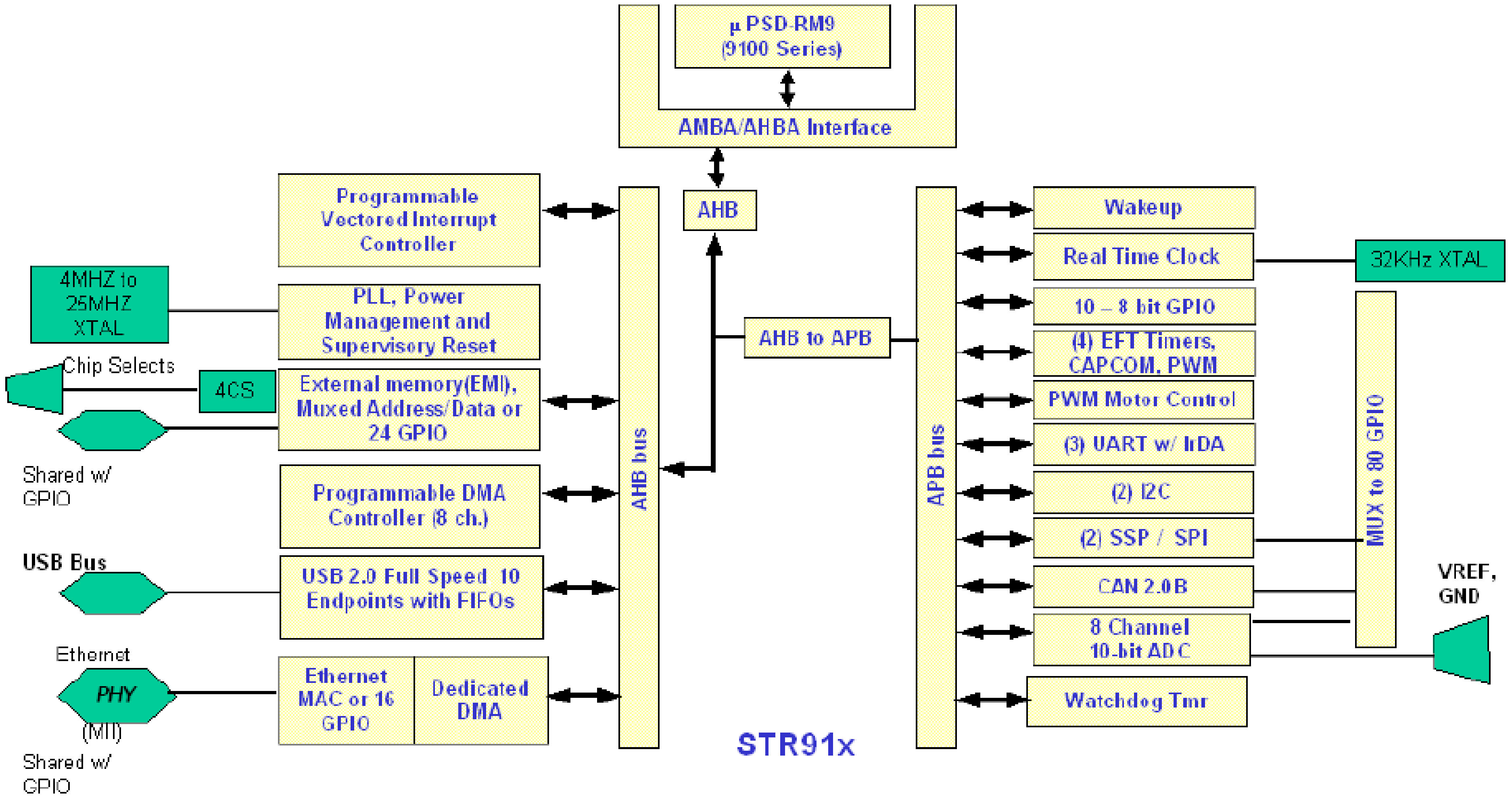
Address offset: 20h

Reset value: 0000 0000h

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							RST_RTC	RST_GPIO9	RST_GPIO8	RST_GPIO7	RST_GPIO6	RST_GPIO5	RST_GPIO4	RST_GPIO3	RST_GPIO2
							rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RST_GPIO1	RST_GPIO0	RST_WIU	Reserved	RST_ADC	RST_CAN	RST_SSP1	RST_SSP0	RST_I2C1	RST_I2C0	RST_UART2	RST_UART1	RST_UART0	RST_MC	RST_TIM23	RST_TIM01
rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

The (bus) architecture of the STR9

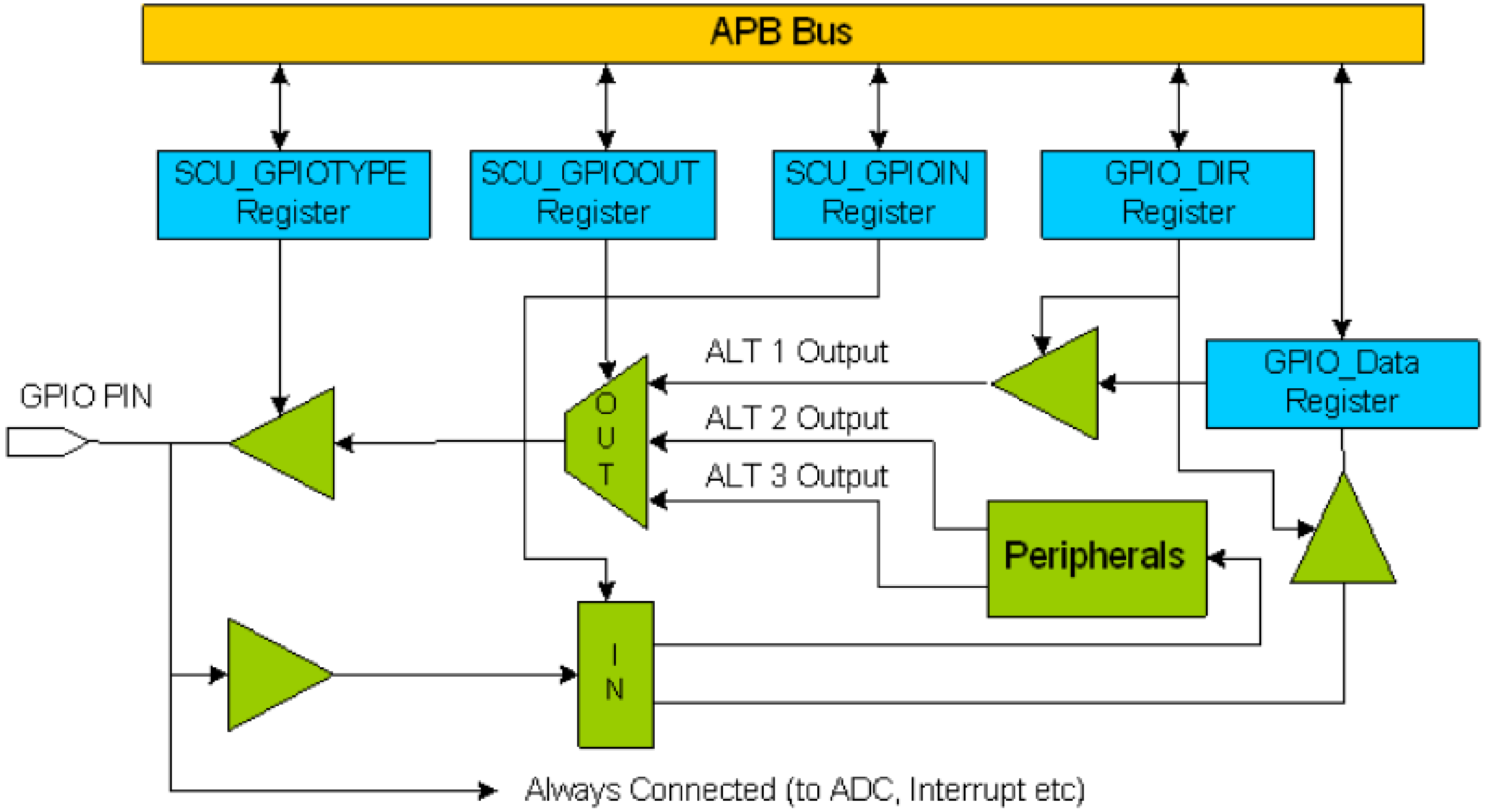


The STR9 has three internal busses. IA high speed bus which connects the CPU to the on-chip memory and complex peripherals the remaining peripherals are connected to two separate peripheral busses.

Arm STR9: General Purpose IO (GPIO)

- GPIO 0 - 9
- All GPIO pins are 5V tolerant.
- I/O port drivers may be configured as push-pull or as open collector.
- Some peripheral functions have bi-directional functionality such as I2C
- data and clock lines.
- Only Port 4 bits 0...7 have an ADC input.
- All ports when configured for Input mode are in high impedance.
- Alternate Input functions default to open and on P0 - P7 are controlled by SCU GPIOIN control registers.
- Alternate Output functions on P0 - P7 are configured via SCU GPIOOUT control registers that select from one of 3 output functions.
- The GPIO ports have no internal or programmable pull-up resistors.

The GPIO

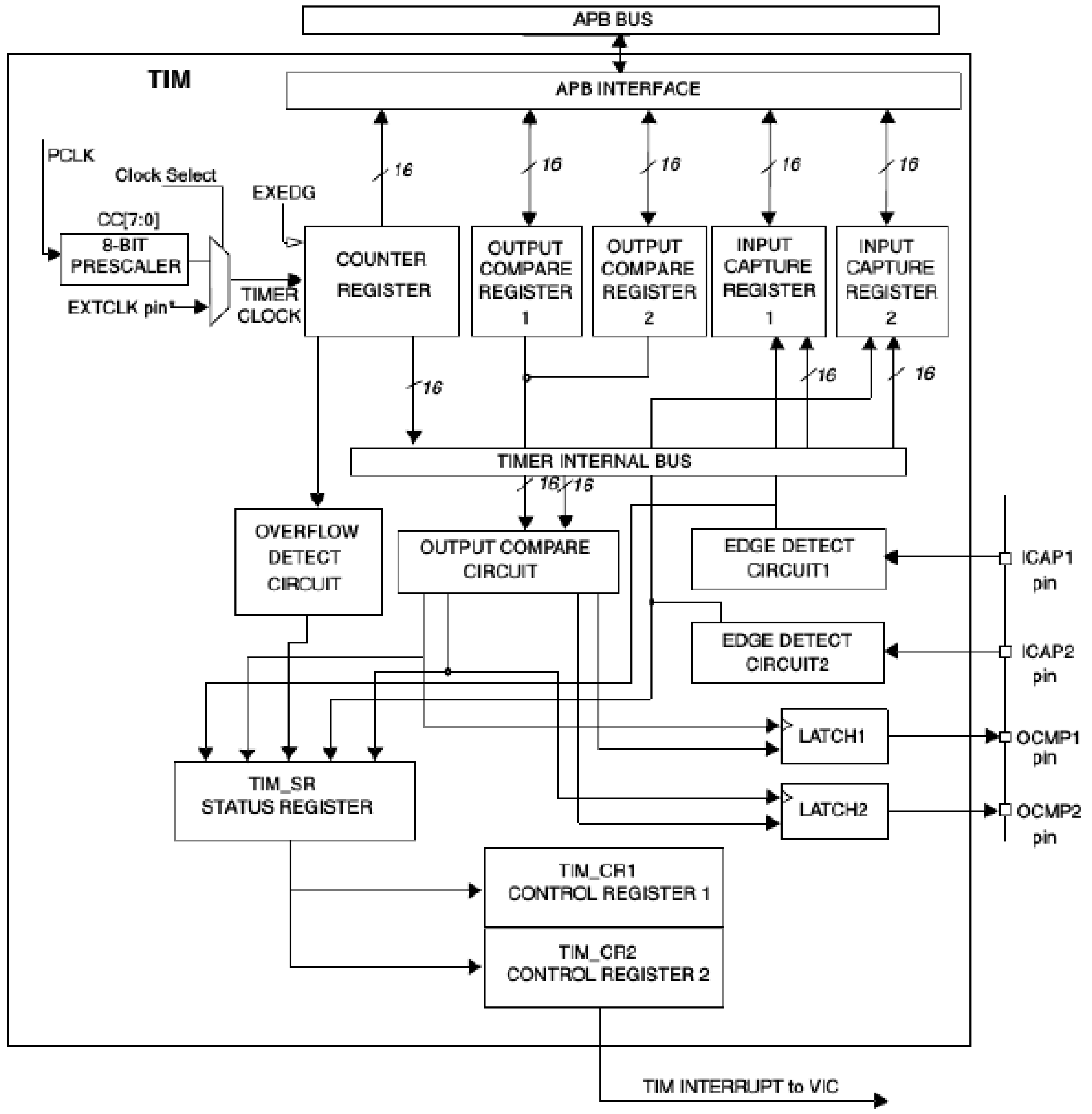


Each STR9 GPIO pin may be configured as input, output or two additional alternate outputs can connect the external pin to on chip peripherals

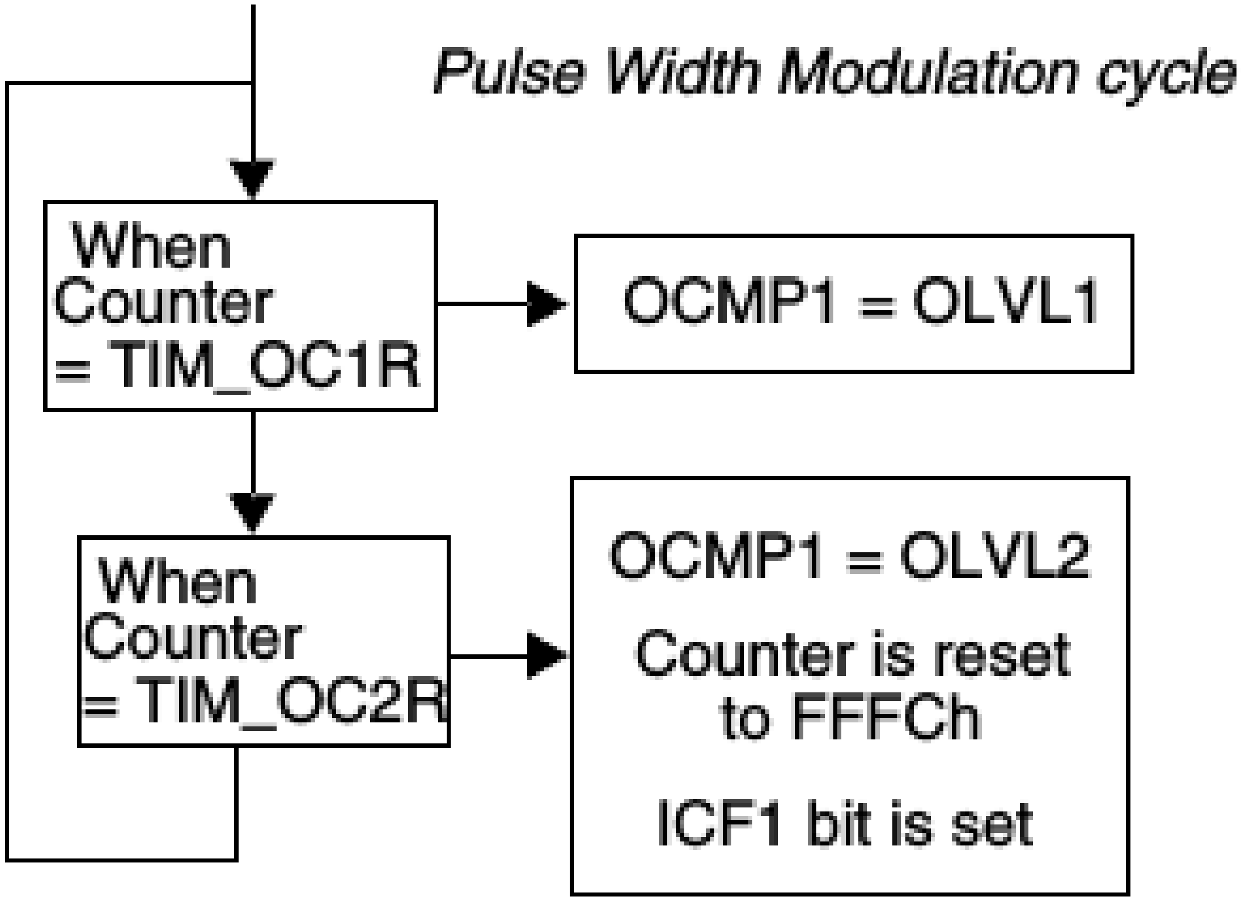
Arm STR9: Timer

- Programmable prescaler: f_{PCLK} divided by 1 to 256 in steps of 1
- Overflow status flag and maskable interrupt
- External clock inputs with the choice of active edge
- Output compare functions with:
  - 2 dedicated 16-bit registers
  - 2 dedicated programmable signals
  - 2 dedicated status flags
  - 1 maskable interrupt
- Input capture functions with:
  - 2 dedicated 16-bit registers
  - 2 dedicated active edge selection signals
  - 2 dedicated status flags
  - 1 maskable interrupt
- Pulse Width Modulation output mode (PWM)
- One Pulse mode (OPM)
- PWM input mode (PWMI)
- 5 alternate functions
- DMA support

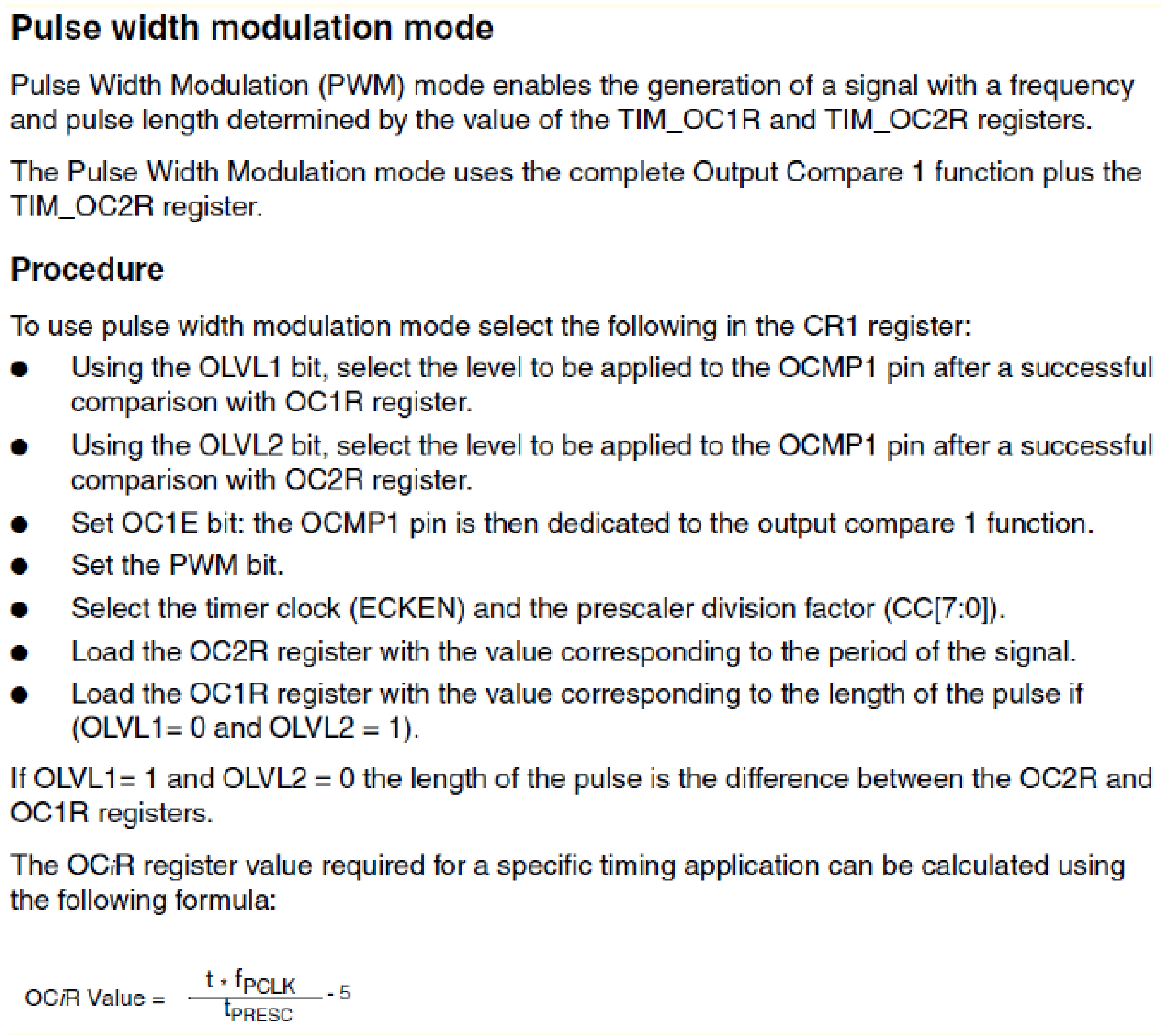
Timer Architecture



PWM


Pulse Width Modulation cycle
When Counter = TIM_OC1R
OCMP1 = OLVL1
When Counter = TIM_OC2R
OCMP1 = OLVL2
Counter is reset to FFFCh
ICF1 bit is set

PWM



Pulse width modulation mode

Pulse Width Modulation (PWM) mode enables the generation of a signal with a frequency and pulse length determined by the value of the TIM_OC1R and TIM_OC2R registers.

The Pulse Width Modulation mode uses the complete Output Compare 1 function plus the TIM_OC2R register.

Procedure

To use pulse width modulation mode select the following in the CR1 register:

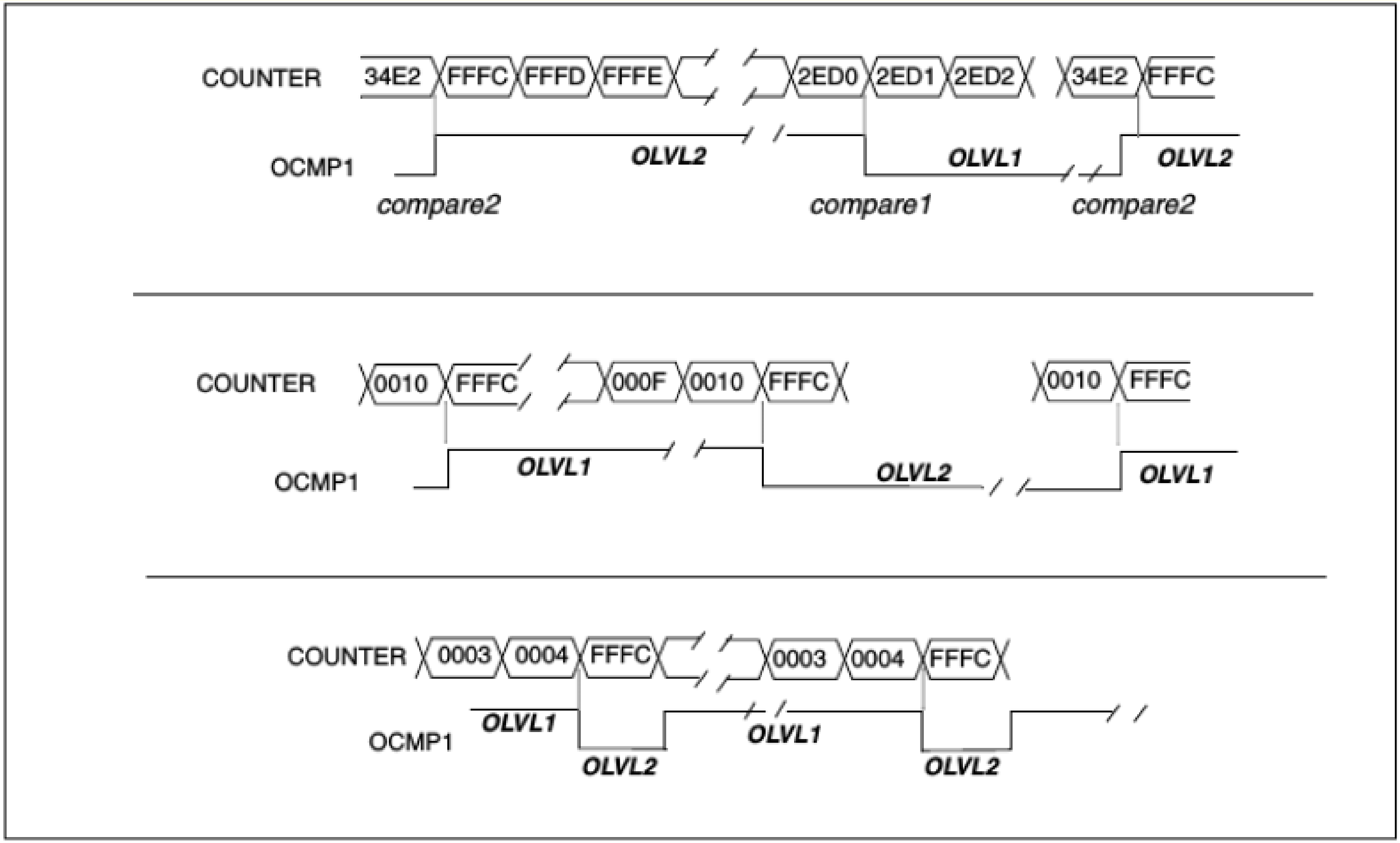
- Using the OLVL1 bit, select the level to be applied to the OCMP1 pin after a successful comparison with OC1R register.
- Using the OLVL2 bit, select the level to be applied to the OCMP1 pin after a successful comparison with OC2R register.
- Set OC1E bit: the OCMP1 pin is then dedicated to the output compare 1 function.
- Set the PWM bit.
- Select the timer clock (ECKEN) and the prescaler division factor (CC[7:0]).
- Load the OC2R register with the value corresponding to the period of the signal.
- Load the OC1R register with the value corresponding to the length of the pulse if (OLVL1= 0 and OLVL2 = 1).

If OLVL1= 1 and OLVL2 = 0 the length of the pulse is the difference between the OC2R and OC1R registers.

The OC*i*R register value required for a specific timing application can be calculated using the following formula:

$$\text{OC}i\text{R Value} = \frac{t \cdot f_{PCLK}}{t_{PRESC}} - 5$$

STR9: PWM: Example



1. In the top part of *Figure 47*, OC1R = 2ED0h, OC2R = 34E2, OLVL1 = 0, OLVL2 = 1.
2. In the middle part of *Figure 47*, OC1R = OC2R = 0010h, OLVL1 = 1, OLVL2 = 0.
3. In the bottom part of *Figure 47*, OC1R = FFFCh, OC2R = 0004h, OLVL1 = 1, OLVL2 = 0.

How to work with STR9

- it is not possible to develop software directly on the microcontroller
- you have to develop the software on a PC
- you have to compile/build the software on a PC
  - therefore you need a so called cross-compiler
- then you have to deploy the executable onto the microcontroller
  - this is called "to flash"

To do this the following tool chain is provided

Tool chain

For the cross compile gcc is used

- the compiler runs on the PC (x86 von Neumann architecture) as the build platform but
- the target platform is a STR9 with ARM7 (Havard) architecture

To build the software we use make

- a makefile is a file that defines in which order all SW-components have to be compiled and have to be linked together
- in the exercises, you just type make in the correct directory
  - where the sources and the make-file is available.

To connect to the chip

- The tool openocd is provided to connect to the chip and to enable to flash the using str9flash.pl

Inline Assembler

part of the exercise code:

```
asm(" \
   MRS r0, cpsr   \n \
   ORR r0, r0, #0x00000040     \n \
   AND r0, r0, #0xffffff7f     \n \
   MRR cpsr_c, r0 \n \
");
```