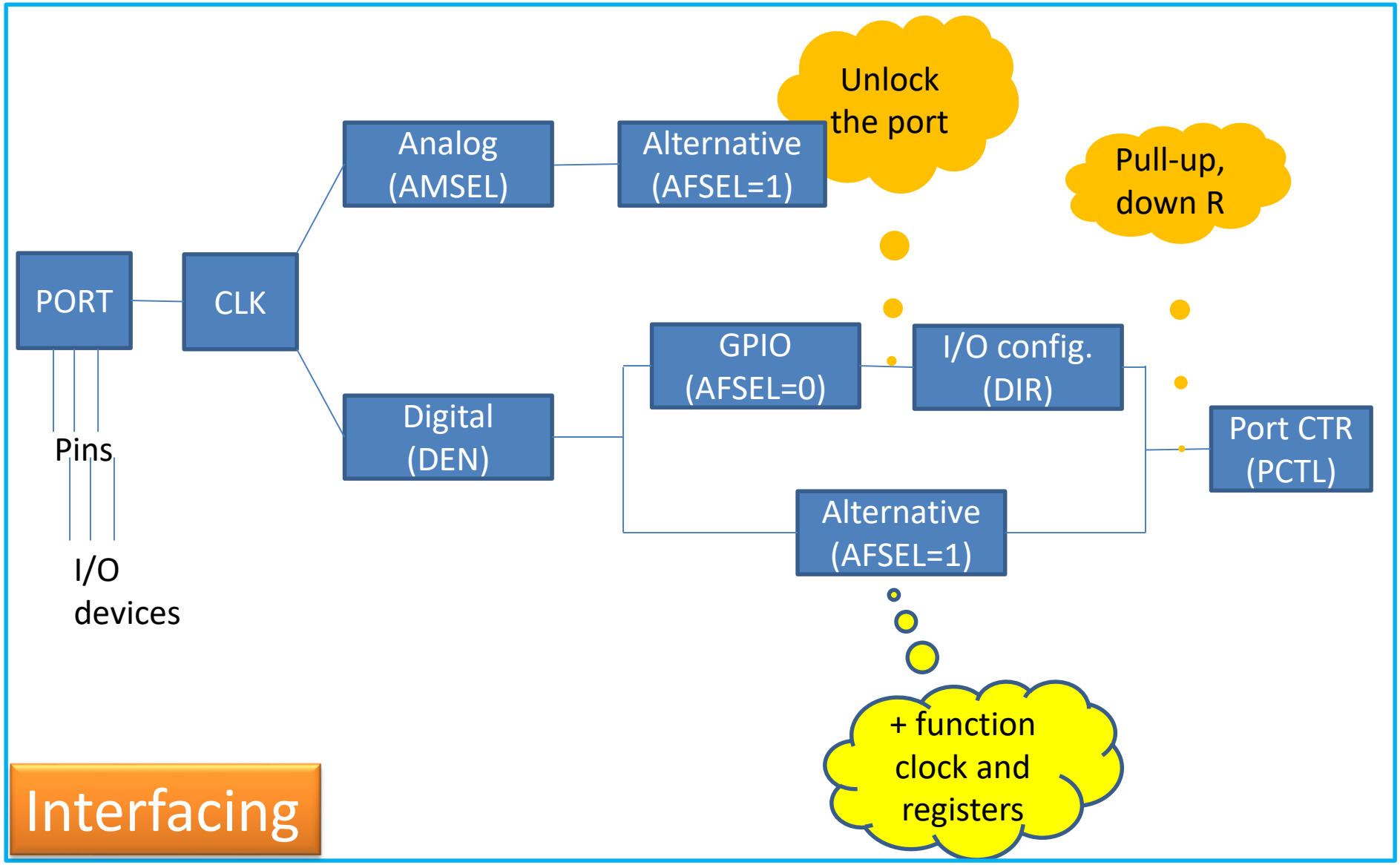
CET 241: Day 9

PLL and SysTick

Dr. Noori Kim

❑ Can you picturize this?



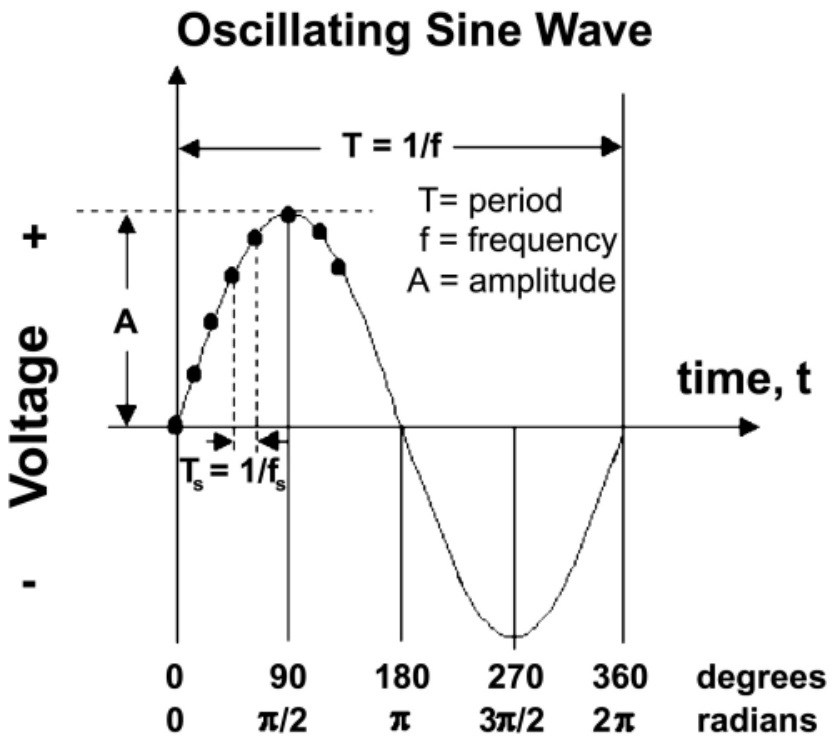
Agenda for today

The focus for the following concept: ‘what’, ‘why’, and ‘how’

1. Phase Lock Loop (PLL)
2. SysTick Timer

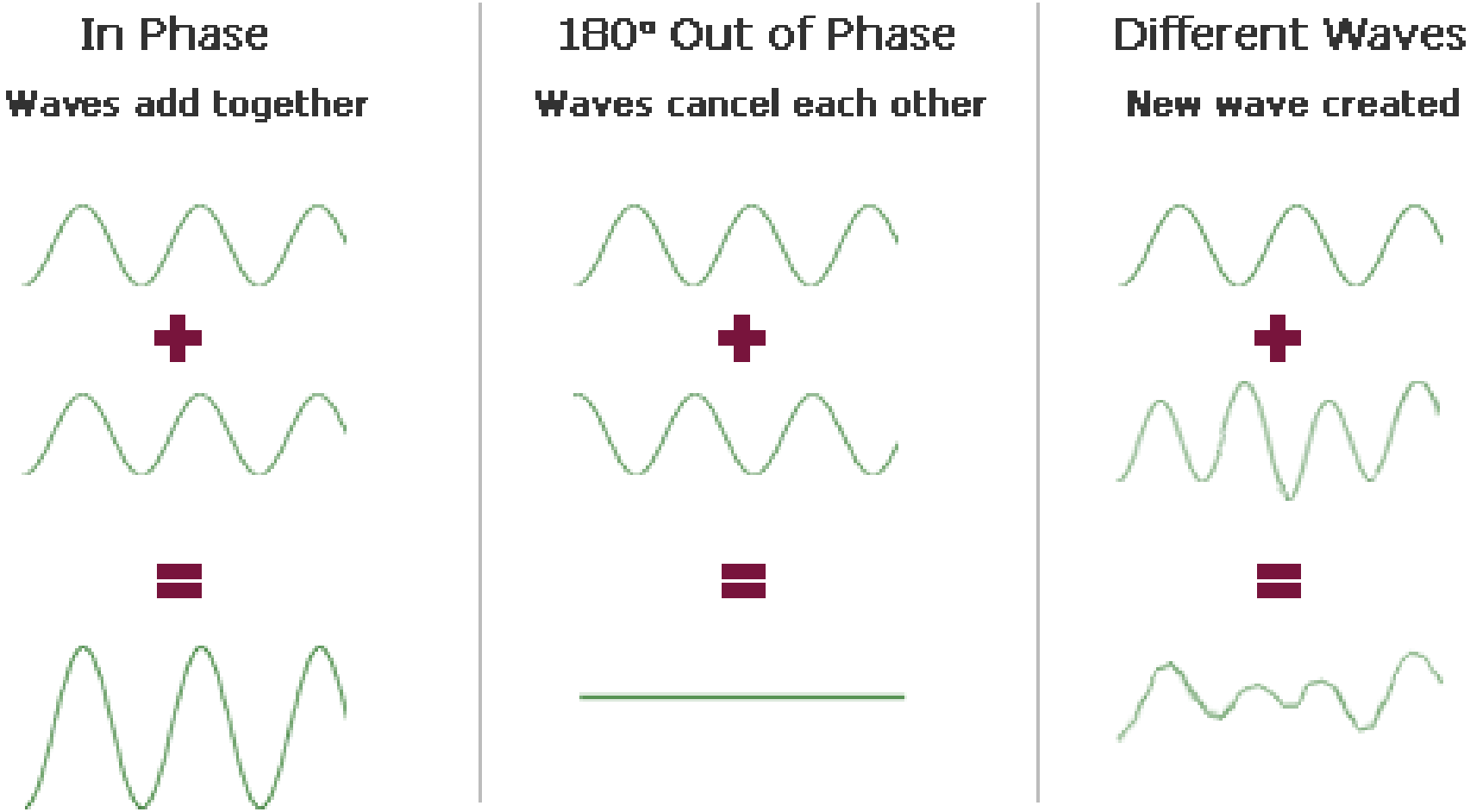
Phase? Lock? And loop?

- What is phase?
  - The position of a point in time (an instant) on a waveform cycle (wiki)



1. T=period=clock period=cycle
2. F=1/T (frequency)
   - Measured in cycles per second
   - Ex:1 kHz = 1 000 cycles / s
   - 1 MHz = 1 000 000 cycles / s
3. A=amplitude
4. Ts=1/fs: Sampling time

Ex) CPU operates at 100 Hz, its "clock cycle" is 0.01 second = 10 ms;



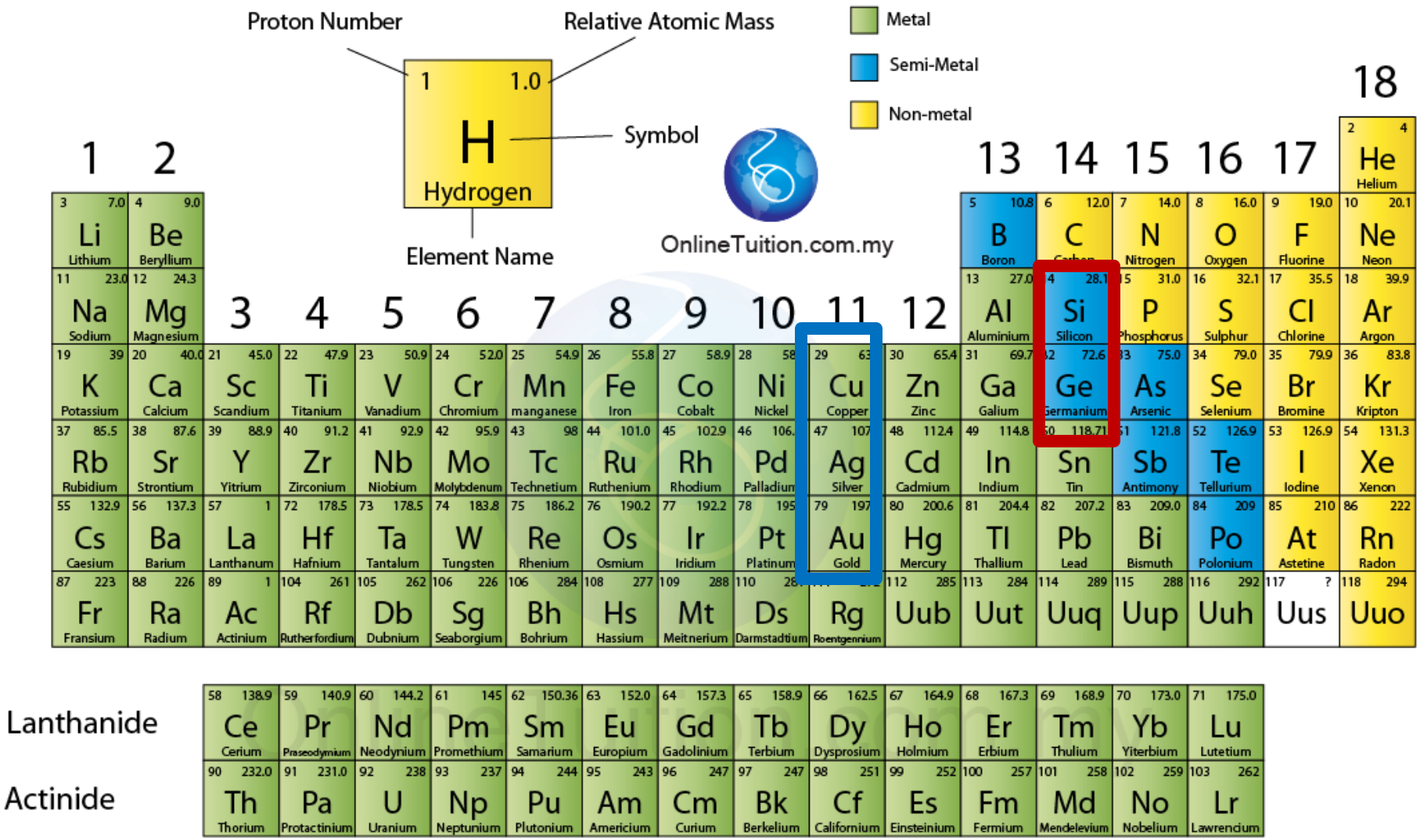
- Phase lock means
  - Forcing to generate in-phase waves by locking their phases
  - But how?

Phase-Lock, and Loop? PLL

- We lock the phase by looping the waves
- That is Phase-Lock-Loop (PLL)
  - Locking phases of clocks (with certain freq.) by looping them until the clocks become in-phase
- Why do we learn this?
  - Most MCUs include PLL
  - A PLL allows a program to adjust **the execution speed** by utilizing **an external crystal**
- What do you mean by **an external crystal**?

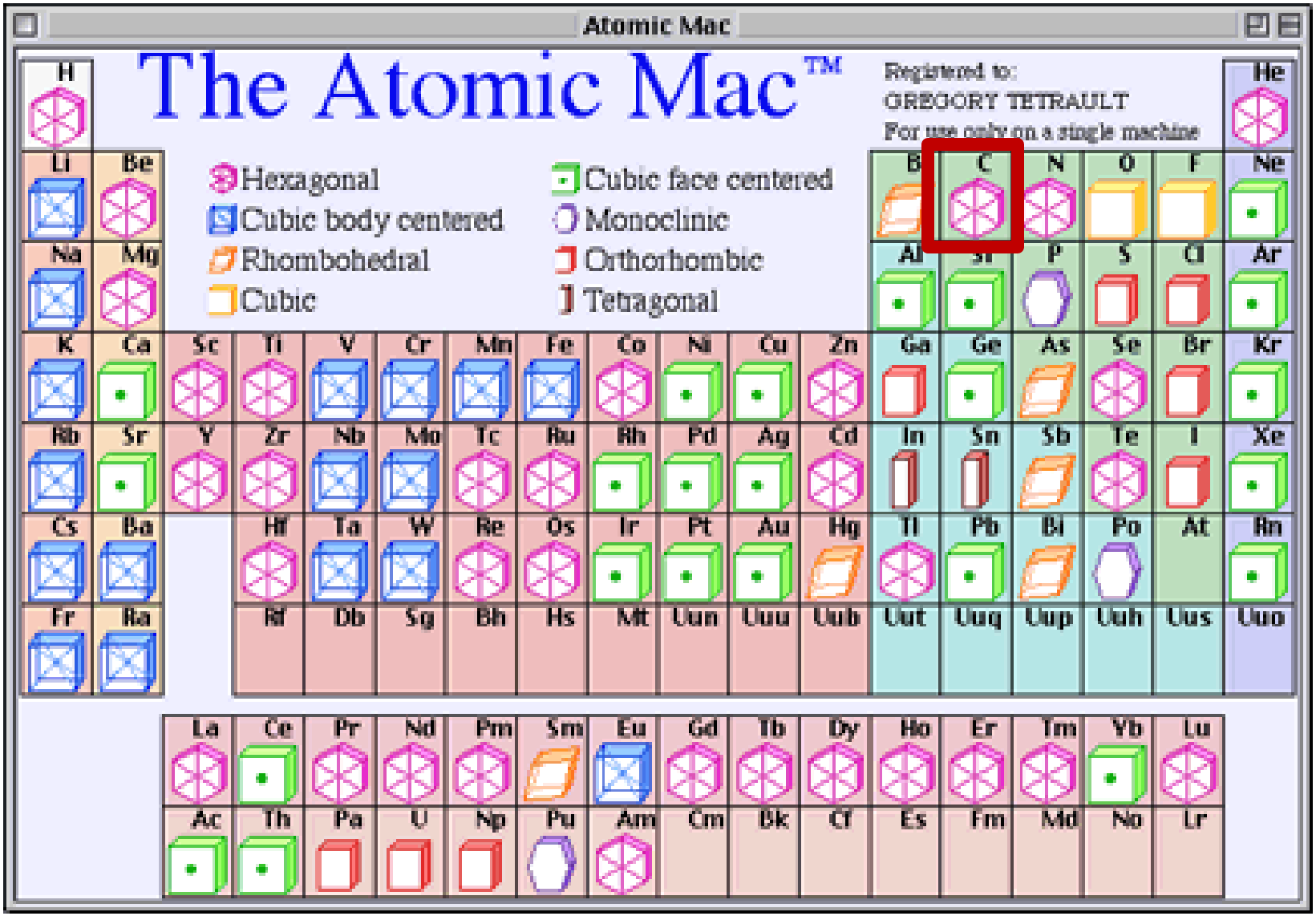
A crystal?

- Computers need a clock, a very accurate clock to run (as a golden reference)
- One can make a clock using semi-conductors: usually they are acceptable but not very accurate
- People found a very accurate clock from the nature, which are crystals



Each element has its own crystal structure

- “Crystal Structure” view of the periodic table



Characteristics of a crystal

- When a voltage is applied across a crystal, it resonates (vibrates) regularly
- Many kinds crystals in the world, each crystal has its own resonance frequency and people utilize this natural characteristic as a clock
- Their resonance behavior as a clock is extremely accurate compared to the other types of oscillators (i.e., semi-conductor osc.)
  - A crystal is used as a golden reference timer

In terms of accuracy

- Ex) Our TM4C123GXL LaunchPad
  - Internal clock (internal oscillator, default clock): 16 MHz ± 1%
    - 16 000 000 cycles/sec, error ± 1%
    - Less precise, less power, no-need external crystal
  - External crystal (main oscillator): ±50 parts per million, about ±5 seconds per day
    - Precise, an option to faster speed (more power), external crystal required

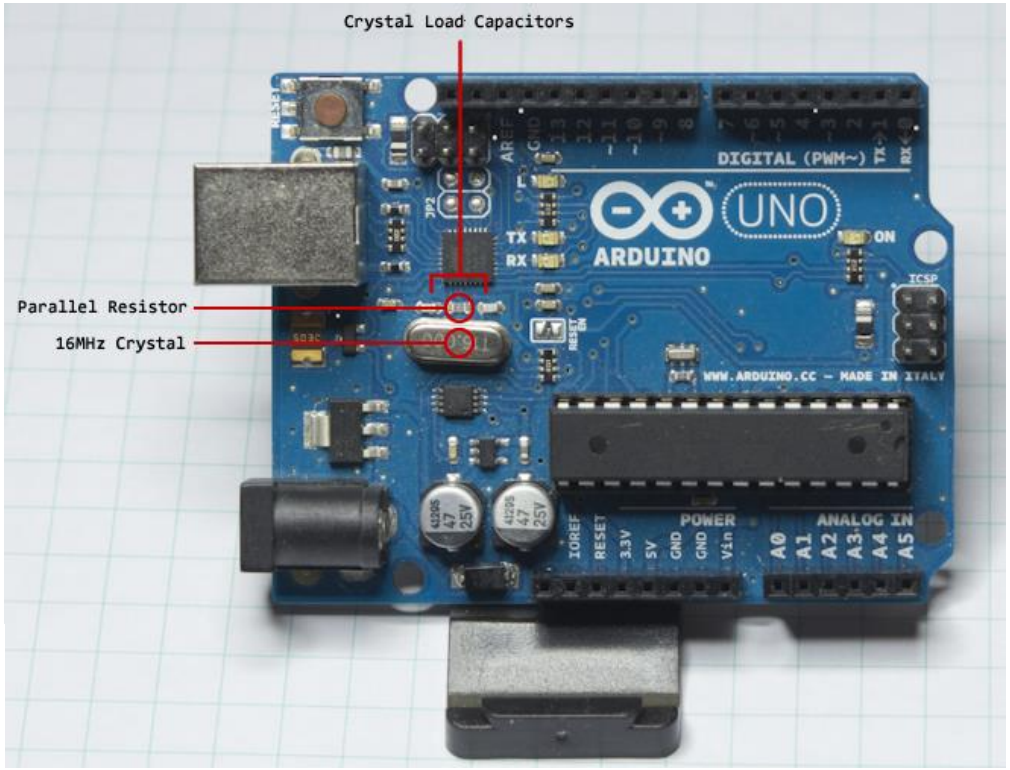
The choice of frequency trade-off

- **Software execution speed** and **electrical power**
  - Slowing down the bus clock: less power to operate and generate less heat
  - Speeding up the bus clock: more calculations per second, at the cost of requiring more power to operate and generating more heat
- When using PLL to speed up, the trade-off must be considered

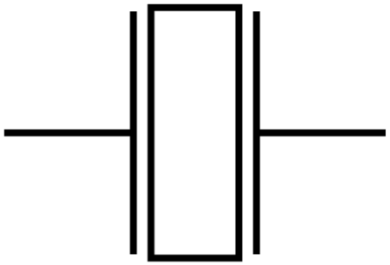
How does it look like?

- Crystal OSC chip

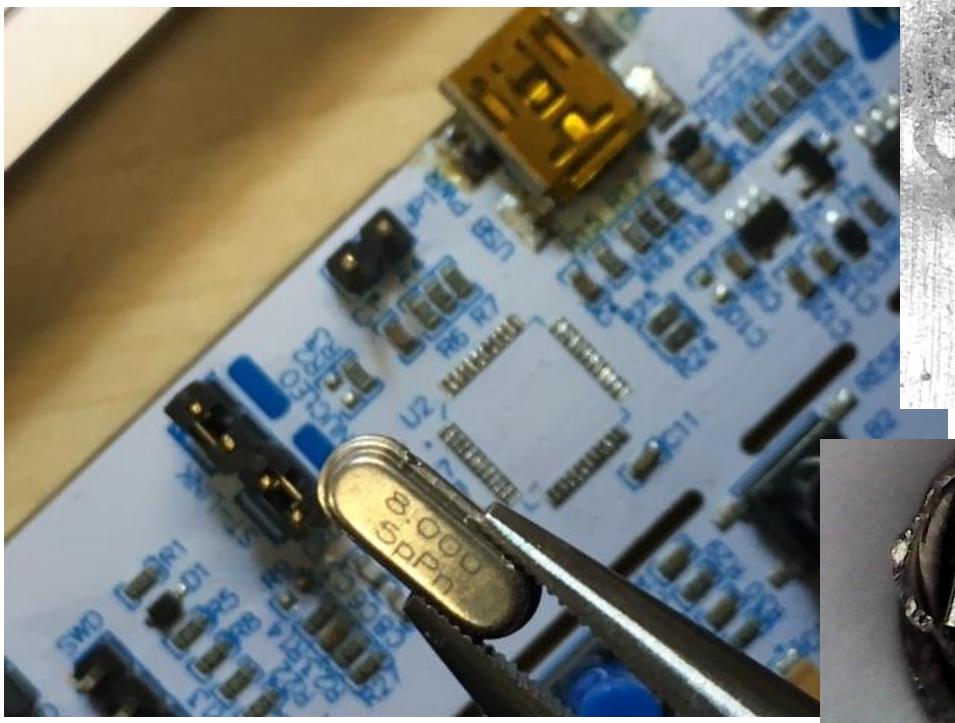
- A crystal In Arduino

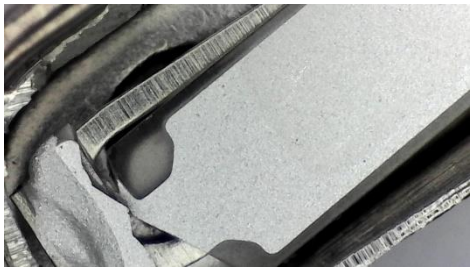


- Symbol

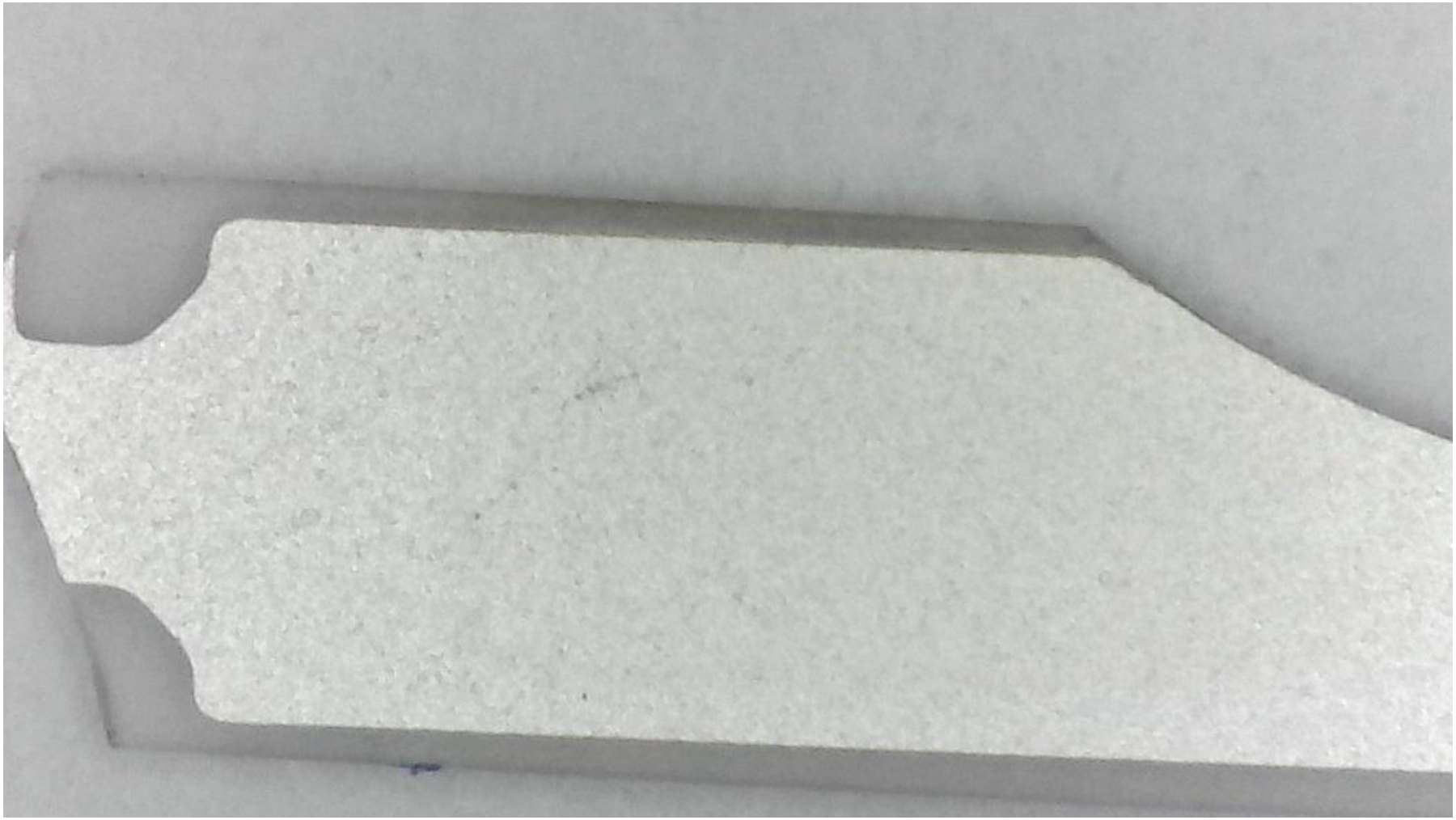


STM32F1 nucleo board’s external osc.



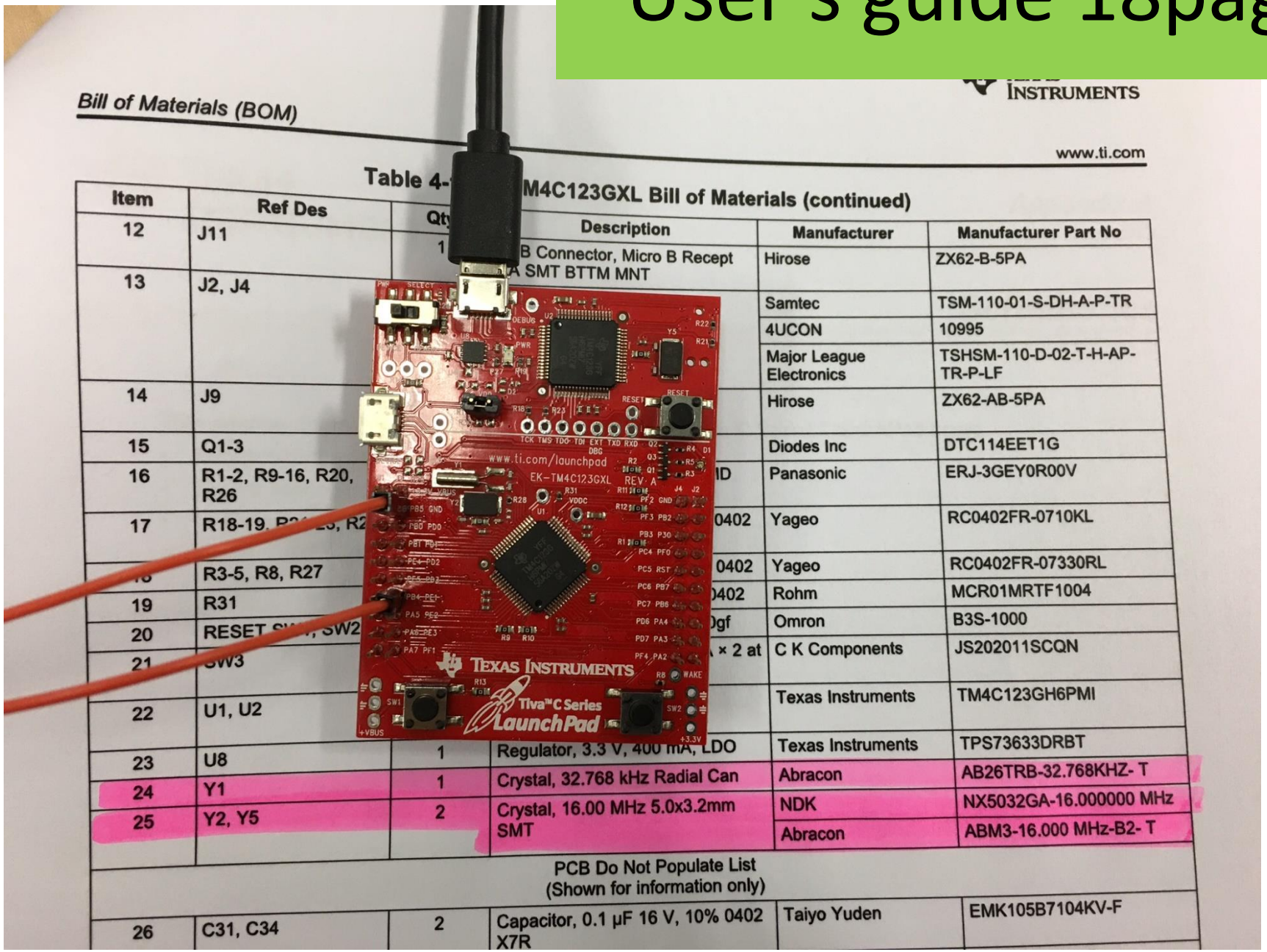


THE CRYSRAL!!!!



Our board

User’s guide 18page

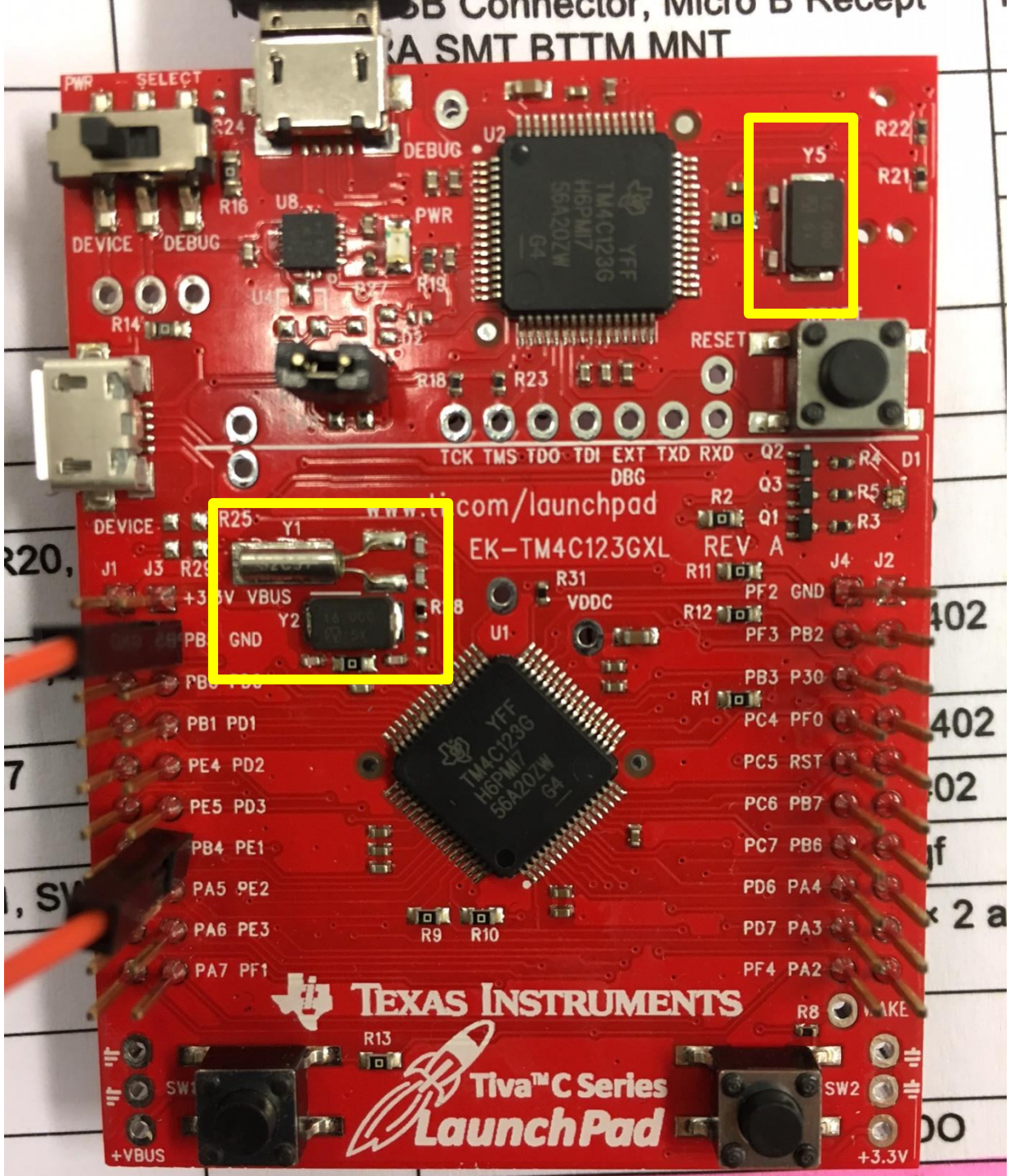


Bill of Materials (BOM)

www.ti.com

Table 4-[illegible] M4C123GXL Bill of Materials (continued)

Item	Ref Des	Qty	Description	Manufacturer	Manufacturer Part No
12	J11	1	[illegible]B Connector, Micro B Recept [illegible]A SMT BTTM MNT	Hirose	ZX62-B-5PA
13	J2, J4			Samtec	TSM-110-01-S-DH-A-P-TR
				4UCON	10995
				Major League Electronics	TSHSM-110-D-02-T-H-AP-TR-P-LF
14	J9			Hirose	ZX62-AB-5PA
15	Q1-3			Diodes Inc	DTC114EET1G
16	R1-2, R9-16, R20, R26		[illegible]ID	Panasonic	ERJ-3GEY0R00V
17	R18-19, R[illegible]		[illegible]0402	Yageo	RC0402FR-0710KL
[illegible]	R3-5, R8, R27		[illegible]0402	Yageo	RC0402FR-07330RL
19	R31		[illegible]0402	Rohm	MCR01MRTF1004
20	RESET [illegible], SW2		[illegible]gf	Omron	B3S-1000
21	SW3		[illegible] × 2 at	C K Components	JS202011SCQN
22	U1, U2			Texas Instruments	TM4C123GH6PMI
23	U8	1	Regulator, 3.3 V, 400 mA, LDO	Texas Instruments	TPS73633DRBT
24	Y1	1	Crystal, 32.768 kHz Radial Can	Abracon	AB26TRB-32.768KHZ- T
25	Y2, Y5	2	Crystal, 16.00 MHz 5.0x3.2mm SMT	NDK	NX5032GA-16.000000 MHz
				Abracon	ABM3-16.000 MHz-B2- T
PCB Do Not Populate List (Shown for information only)					
26	C31, C34	2	Capacitor, 0.1 µF 16 V, 10% 0402 X7R	Taiyo Yuden	EMK105B7104KV-F


PWR
SELECT
DEVICE
DEBUG
R24
R16
U8
U4
R14
DEBUG
PWR
U2
R19
R18
R23
Y5
R22
R21
RESET
TCK TMS TDO TDI EXT TXD RXD
DBG
Q2
Q3
Q1
R4
R5
R3
D1
R2
com/launchpad
EK-TM4C123GXL
REV A
DEVICE
R25
Y1
Y2
VBUS
GND
J1
J3
R29
R31
VDDC
U1
R11
R12
R1
J4
J2
PF2 GND
PF3 PB2
PB3 P30
PC4 PF0
PC5 RST
PC6 PB7
PC7 PB6
PD6 PA4
PD7 PA3
PF4 PA2
PB1 PD1
PE4 PD2
PE5 PD3
PB4 PE1
PA5 PE2
PA6 PE3
PA7 PF1
R9
R10
R8
R13
TEXAS INSTRUMENTS
Tiva™C Series
LaunchPad
SW1
SW2
+VBUS
+3.3V

Tools | tm4c123gh6pm Da... | ST7735R_V0.2-TFT. | LaunchPadUsersM... ×

12 / 27 100%

TEXAS INSTRUMENTS

In-Circuit Debug Interface (ICDI)

There is no external battery source on the Tiva C Series LaunchPad Hibernation modu
the VDD3ON power control mechanism should be used. This mechanism uses interna
remove power from the Cortex-M4 processor as well as to most analog and digital fun
retaining I/O pin power.

To measure the Hibernation mode current or the Run mode current, the VDD jumper t
V pin and the MCU_PWR pin must be removed. See the complete schematics (appen
document) for details on these pins and component locations. An ammeter should ther
between the 3.3 V pin and the MCU_PWR pin to measure I_{DD} (or I_{HIB_VDD3ON}). The TM4
microcontroller uses V_{DD} as its power source during V_{DD3ON} Hibernation mode, so I_{DD} is
mode (VDD3ON mode) current. This measurement can also be taken during Run mod
I_{DD} the microcontroller running current.

2.2.3 Clocking

The Tiva C Series LaunchPad uses a 16.0-MHz crystal (Y2) to complete the TM4C123GH6PM microcontroller main internal clock circuit. An internal PLL, configured in software, multiples this clock to higher frequencies for core and peripheral timing.

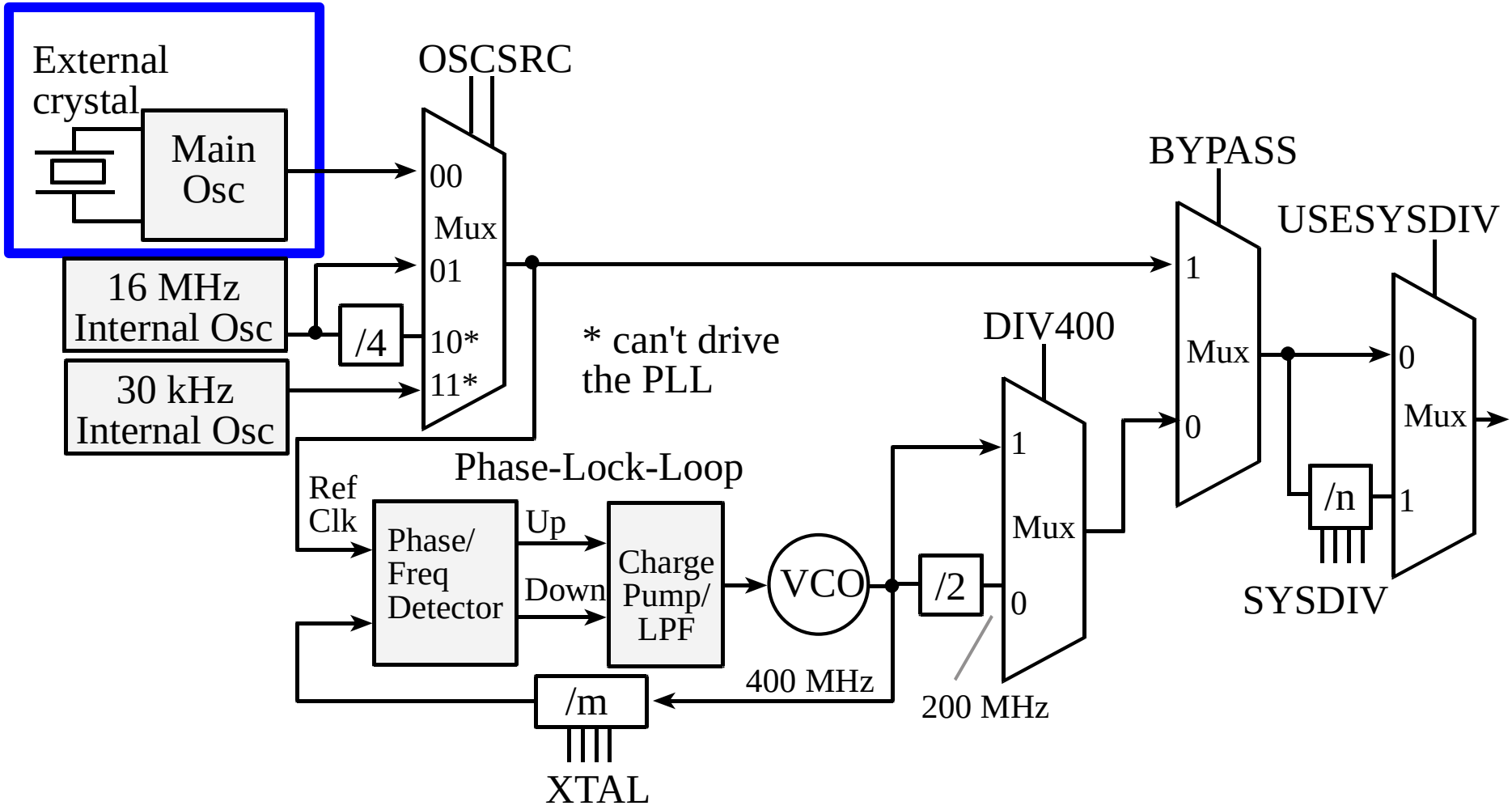
The Hibernation module is clocked from an external 32.768-KHz crystal (Y1).

2.2.2 Hibernate

The Tiva C Series LaunchPad provides an external 32.768-kHz crystal (Y1) as the clock source for the TM4C123GH6PM Hibernation module clock source. The current draw while in Hibernate mode can be measured by making some minor adjustments to the Tiva C Series LaunchPad. This procedure is explained in more detail later in this section.

The conditions that can generate a wake signal to the Hibernate module on the Tiva C Series LaunchPad are waking on a Real-time Clock (RTC) match and/or waking on assertion of the $\overline{\text{WAKE}}$ pin. [1] The second user switch (SW2) is connected to the $\overline{\text{WAKE}}$ pin on the microcontroller. The $\overline{\text{WAKE}}$ pin, as well as the V_{DD} and $\overline{\text{HIB}}$ pins, are easily accessible through breakout pads on the Tiva C Series LaunchPad. See the appended schematics for details.

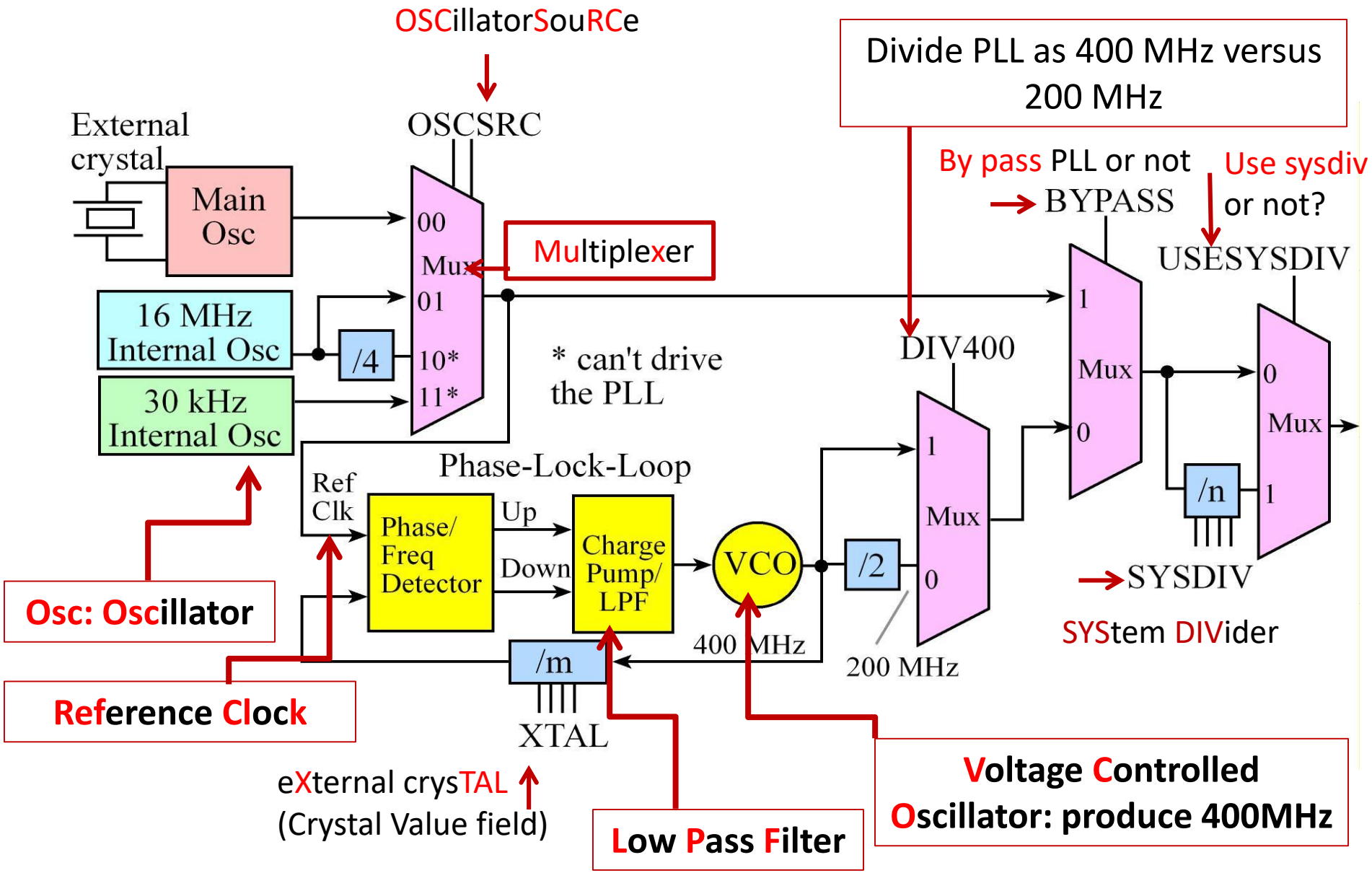
Phase Lock Loop (PLL) of our board

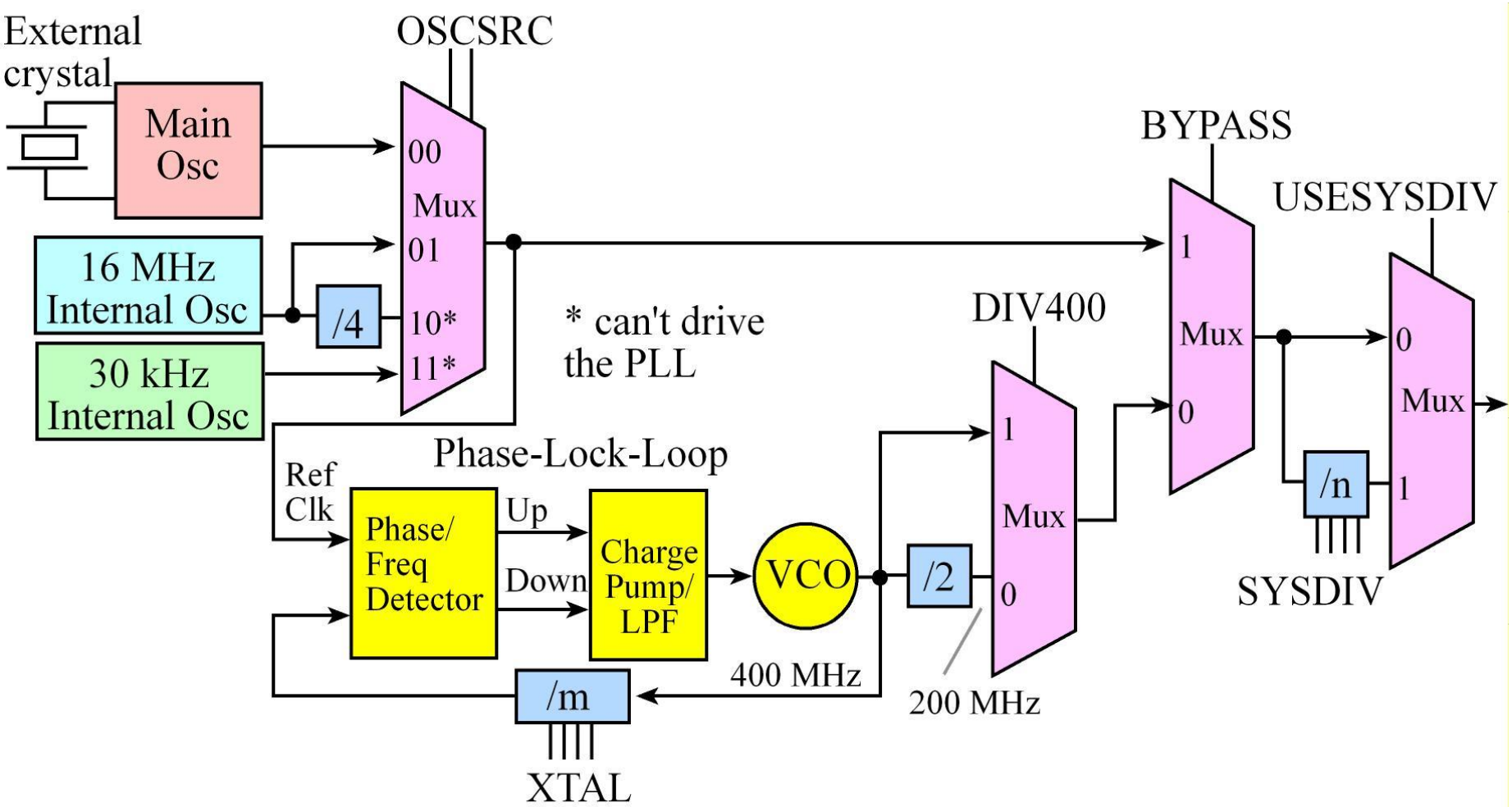


Need to be familiarized w/ the names first

1) Osc: Oscillator
2) OSCSRC: OSCillatorSouRCe
3) MUX: Multiplexer
4) RefClk: Reference Clock
5) XTAL: eXternal crysTAL (Crystal Value field)
6) LPF: Low Pass Filter

7) VCO: Voltage Controlled Oscillator: produce 400MHz
8) SYSDIV: SYStem DIVider
9) DIV400: DIVide PLL as 400 MHz versus 200 MHz
10) BYPASS: By pass PLL or not
11) USESYSDIV: Use sysdiv or not?


OSCillatorSouRCe
OSCSRC
Divide PLL as 400 MHz versus 200 MHz
By pass PLL or not
BYPASS
Use sysdiv or not?
USESYSDIV
External crystal
Main Osc
00
Mux
01
10*
11*
Multiplexer
16 MHz Internal Osc
/4
30 kHz Internal Osc
* can't drive the PLL
DIV400
1
Mux
0
Phase-Lock-Loop
Ref Clk
Phase/ Freq Detector
Up
Down
Charge Pump/ LPF
VCO
/2
/n
SYSDIV
SYStem DIVider
Osc: Oscillator
Reference Clock
/m
400 MHz
200 MHz
XTAL
eXternal crysTAL
(Crystal Value field)
Low Pass Filter
Voltage Controlled Oscillator: produce 400MHz



- 3 types of OSCs: main, internal, VCO
  - Internal osc: minimal power but is imprecise
  - External crystal: stable bus clock (TM4C123: equipped with 16 MHz crystal) but needs more power

PLL related registers (main clock R)

Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	30	28-22	13	11	6-4	
$400FE070	USERCC2	DIV400	SYSDIV2	PWRDN2	BYPASS2	OSCSRC2	SYSCTL_ RCC2_R

- Run mode Clock Configure
- Raw Interrupt Status

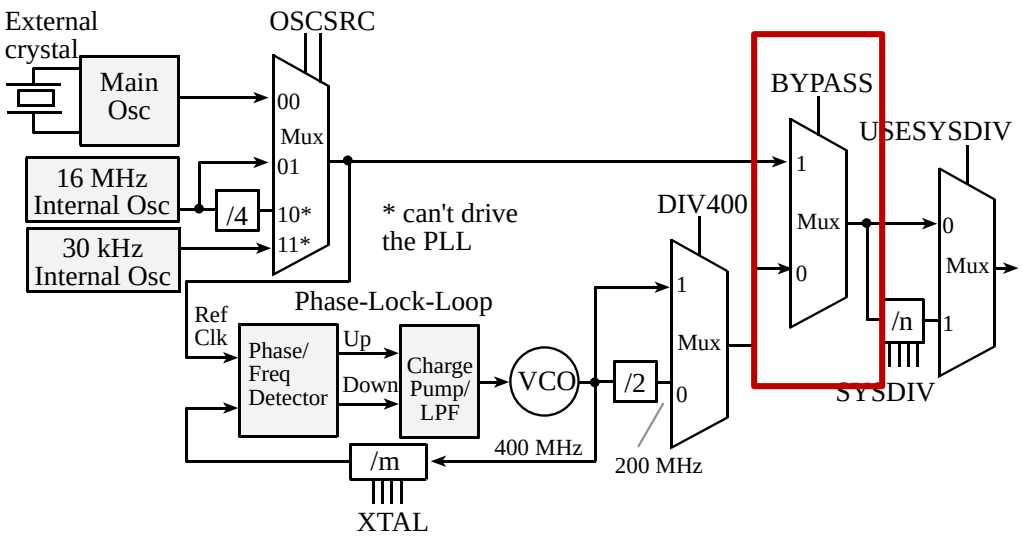
- Let’s do the PLL Initialization ☺

Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	30	28-22	13	11	6-4	
$400FE070	**USERCC 2**	DIV400	SYSDIV2	PWRDN2	BYPASS2	OSCSRC2	**SYSCTL_ RCC2_R**

0) Use RCC2 because it provides for more options.

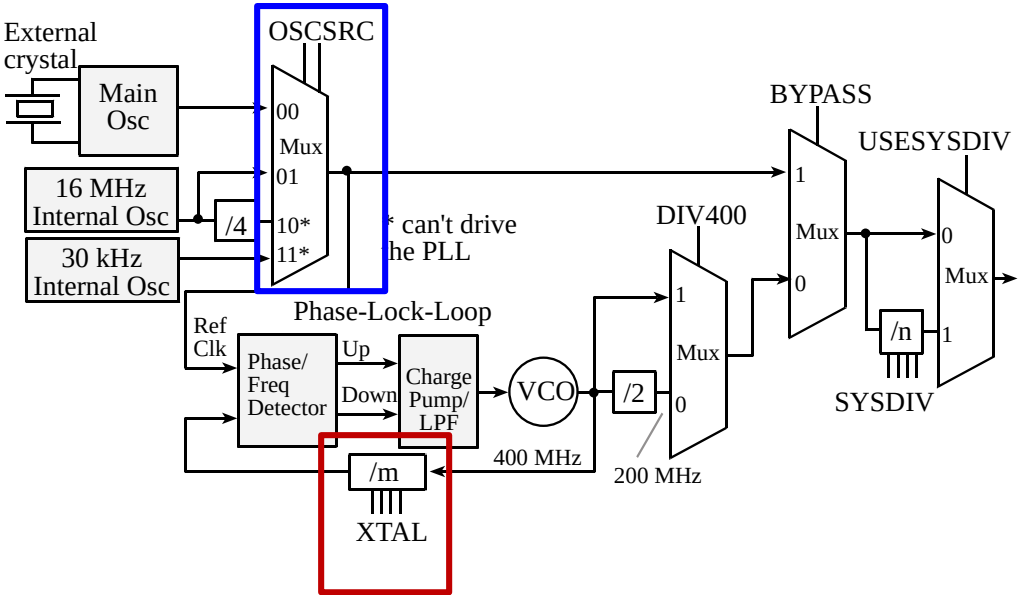
Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	30	28-22	13	**11**	6-4	
$400FE070	USERCC2	DIV400	SYSDIV2	PWRDN2	**BYPASS2**	OSCSRC2	**SYSCTL_ RCC2_R**

1) The first step is to set BYPASS2 (bit 11). At this point the PLL is bypassed and there is no system clock divider.



Address	26-23	22	13	11	**10-6**	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	**XTAL**	OSCSRC	**SYSCTL_ RCC_R**
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	30	28-22	13	11	**6-4**	
$400FE070	USERCC2	DIV400	SYSDIV2	PWRDN2	BYPASS2	**OSCSRC2**	**SYSCTL_ RCC2_R**

2) The second step is to specify the crystal frequency in the five XTAL bits. The OSCSRC2 bits are cleared to select the main oscillator as the oscillator clock source.

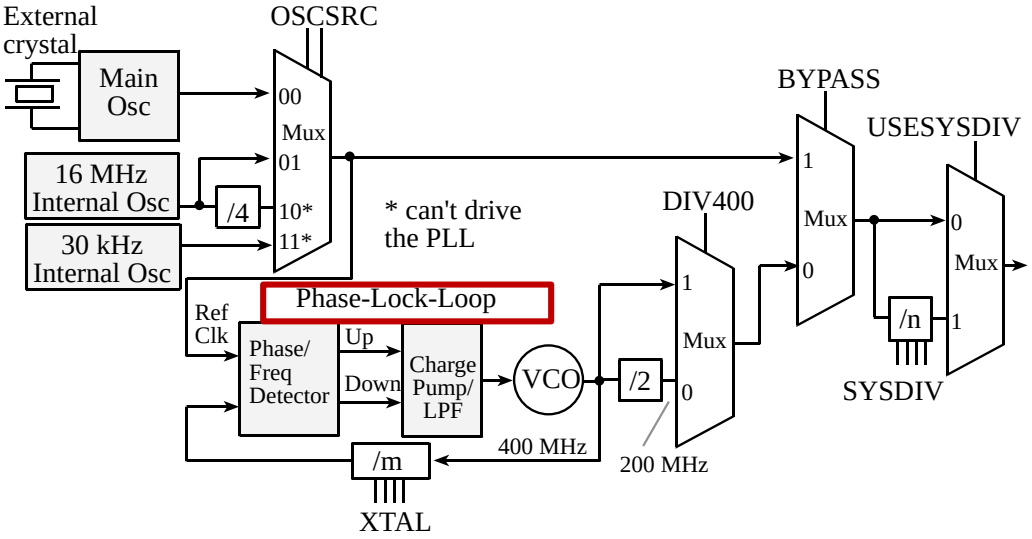


XTAL bits code

XTAL	*Crystal Freq (MHz)*	*XTAL*	*Crystal Freq (MHz)*
0x0	Reserved	0x10	10.0 MHz
0x1	Reserved	0x11	12.0 MHz
0x2	Reserved	0x12	12.288 MHz
0x3	Reserved	0x13	13.56 MHz
0x4	3.579545 MHz	0x14	14.31818 MHz
0x5	3.6864 MHz	**0x15**	**16.0 MHz**
0x6	4 MHz	0x16	16.384 MHz
0x7	4.096 MHz	0x17	18.0 MHz
0x8	4.9152 MHz	0x18	20.0 MHz
0x9	5 MHz	0x19	24.0 MHz
0xA	5.12 MHz	0x1A	25.0 MHz
0xB	6 MHz (reset value)	0x1B	Reserved
0xC	6.144 MHz	0x1C	Reserved
0xD	7.3728 MHz	0x1D	Reserved
0xE	8 MHz	0x1E	Reserved
0xF	8.192 MHz	0x1F	Reserved

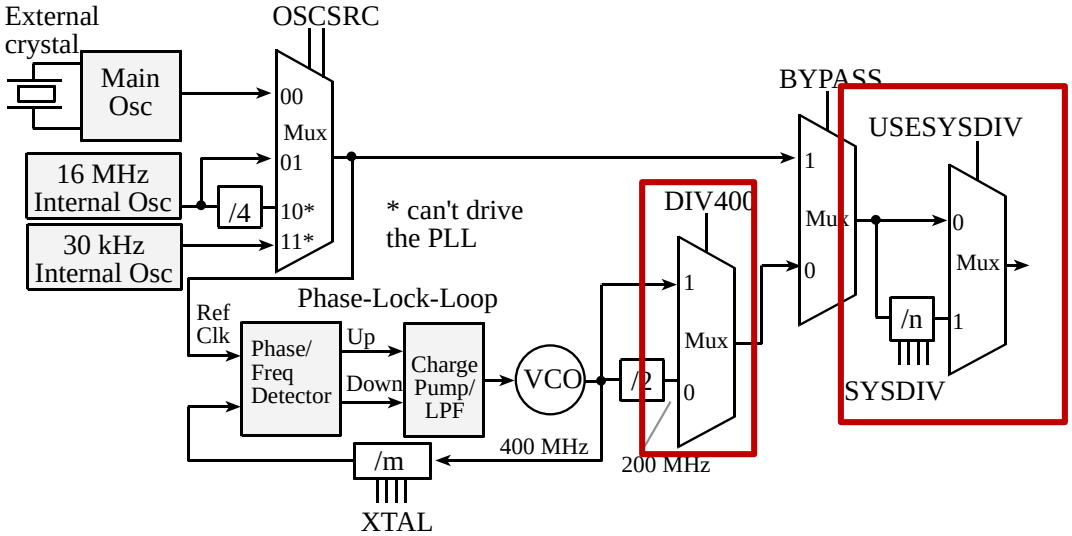
Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	30	28-22	**13**	11	6-4	
$400FE070	USERCC2	DIV400	SYSDIV2	**PWRDN2**	BYPASS2	OSCSRC2	**SYSCTL_ RCC2_R**

3) The third step is to clear PWRDN2 (bit 13, PLL power down: 0 PLL on, 1: PLL power down) to activate the PLL.



Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	**30**	**28-22**	13	11	6-4	
$400FE070	USERCC2	**DIV400**	**SYSDIV2**	PWRDN2	BYPASS2	OSCSRC2	**SYSCTL_ RCC2_R**

4) The fourth step is to set the desired system divider ➔use 400 MHz PLL (DIV400 field) And to configure and enable the clock divider using the 7-bit SYSDIV2 field. If the 7-bit SYSDIV2 is **n**, then the clock will be divided by **n**+1.



i.e.) To get the desired 80 MHz from the 400 MHz PLL, we need to divide by 5. So, we place a 4 into the SYSDIV2 field.

Page 260

Run-Mode Clock Configuration 2 (RCC2)

Base 0x400F.E000
Offset 0x070
Type RW, reset 0x07C0.6810

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	USERCC2	DIV400	reserved	SYSDIV2						SYSDIV2LSB	reserved					
Type	RW	RW	RO	RW	RW	RW	RW	RW	RW	RW	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved	USBPWRDN	PWRDN2	reserved	BYPASS2	reserved				OSCSRC2			reserved			
Type	RO	RW	RW	RO	RW	RO	RO	RO	RO	RW	RW	RW	RO	RO	RO	RO
Reset	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31	USERCC2	RW	0	Use RCC2 Value Description 0 The RCC register fields are used, and the fields in RCC2 are ignored. 1 The RCC2 register fields override the RCC register fields.
30	DIV400	RW	0	Divide PLL as 400 MHz versus 200 MHz This bit, along with the SYSDIV2LSB bit, allows additional frequency choices. Value Description

Bit/Field	Name	Type	Reset	Description
29	reserved	RO	0x0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
28:23	SYSDIV2	RW	0x0F	System Clock Divisor 2 Specifies which divisor is used to generate the system clock from either the PLL output or the oscillator source (depending on how the BYPASS2 bit is configured). SYSDIV2 is used for the divisor when both the USESYSDIV bit in the RCC register and the USERCC2 bit in this register are set. See Table 5-5 on page 223 for programming guidelines.
22	SYSDIV2LSB	RW	1	Additional LSB for SYSDIV2 When DIV400 is set, this bit becomes the LSB of SYSDIV2. If DIV400 is clear, this bit is not used. See Table 5-5 on page 223 for programming guidelines. This bit can only be set or cleared when DIV400 is set.

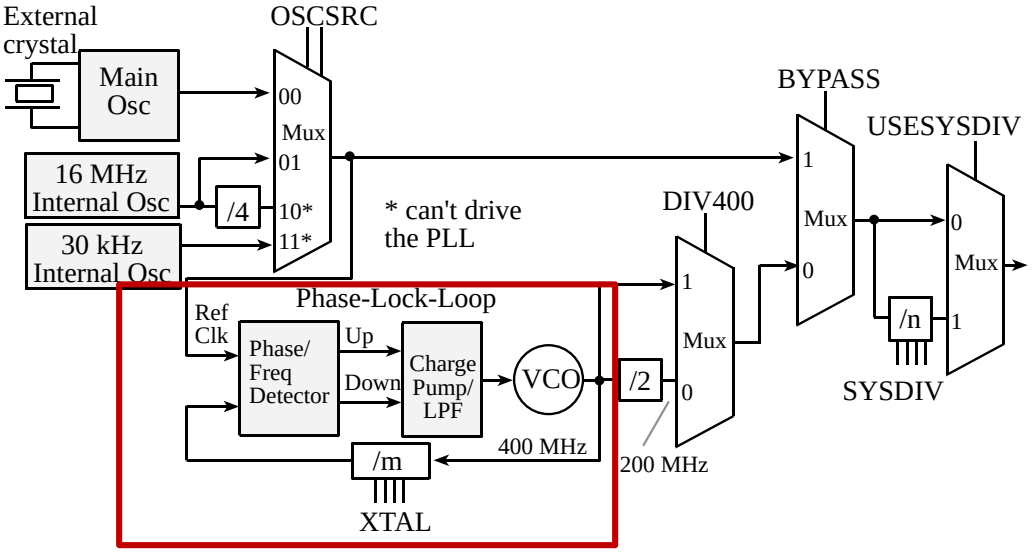
Example SYSDIV2 (n) values:

- n=4 gives 400/(4+1) = 80 MHz (max)
- n=7 gives 400/(7+1) = 50MHz
- n=9 gives 400/(9+1) = 40MHz
- n=15 gives 400/(15+1) = 25MHz

A sidebar: +1 thing always happens when we deal with programming related calculation. Why?

Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		**SYSCTL_ RIS_R**
	31	30	28-22	13	11	6-4	
$400FE070	USERCC2	DIV400	SYSDIV2	PWRDN2	BYPASS2	OSCSRC2	SYSCTL_ RCC2_R

5) The fifth step is to wait for the PLL to stabilize by waiting for PLLRIS (bit 6, PLL Raw Interrupt Status) in the **SYSCTL_RIS_R** to become high.

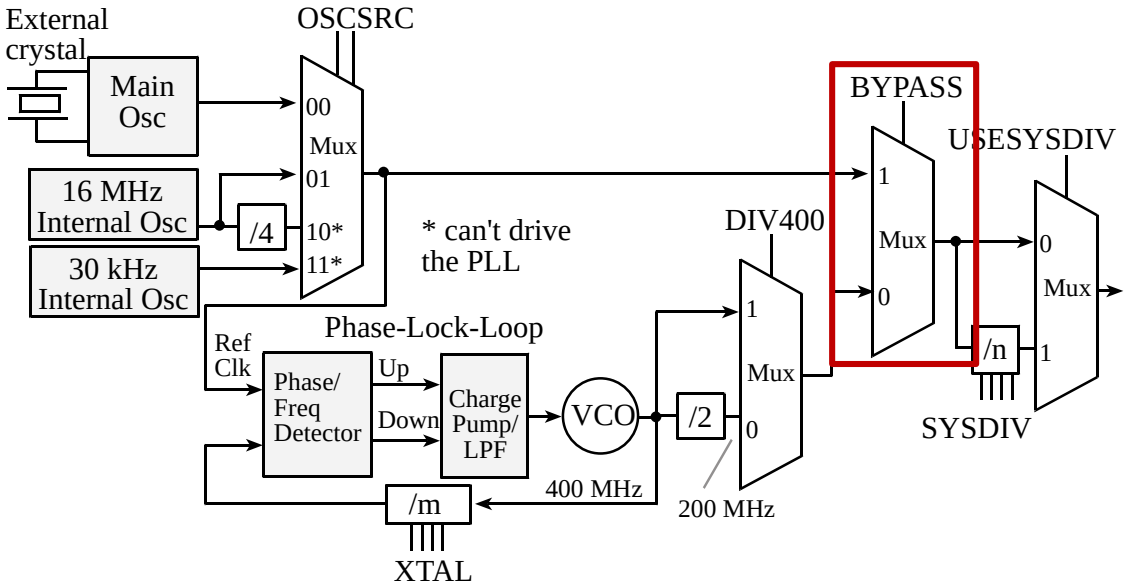


In general, we can solve 2 problems using PLL

1. Inaccuracy of clock: by referencing a golden clock
2. Clock edge (rising or falling) matching between identical clocks

Address	26-23	22	13	11	10-6	5-4	Name
$400FE060	SYSDIV	USESYSDIV	PWRDN	BYPASS	XTAL	OSCSRC	SYSCTL_ RCC_R
$400FE050					PLLRIS		SYSCTL_ RIS_R
	31	30	28-22	13	**11**	6-4	
$400FE070	USERCC2	DIV400	SYSDIV2	PWRDN2	**BYPASS2**	OSCSRC2	**SYSCTL_ RCC2_R**

6) The last step is to connect the PLL by clearing the BYPASS2 bit.



- To modify this program to operate on other microcontrollers, you will need to change the **crystal frequency** and the system clock divider.

```
void PLL_Init(void){
    // 0) Use RCC2
    SYSCTL_RCC2_R |=  0x80000000;  // USERCC2 (31st bit of RCC2)
    // 1) bypass PLL while initializing
    SYSCTL_RCC2_R |=  0x00000800;  // BYPASS2, PLL bypass
    // 2) select the crystal value and oscillator source
    SYSCTL_RCC_R = (SYSCTL_RCC_R &~0x000007C0)   // clear XTAL field, bits 10-6
            + (0x15<<6);   // 101.0100.0000, configure for 16 MHz crystal 0x00000540
    SYSCTL_RCC2_R &= ~0x00000070;  // configure for main oscillator source (4th -6th bit:00)
    // 3) activate PLL by clearing PWRDN
    SYSCTL_RCC2_R &= ~0x00002000;
    // 4) set the desired system divider
    SYSCTL_RCC2_R |= 0x40000000;   // use 400 MHz PLL (DIV400 field)
    SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~ 0x1FC00000)  // clear system clock divider
            + (4<<22);     // configure for 80 MHz clock (400/5=80)
    // 5) wait for the PLL to lock by polling PLLRIS (phase lock, phase stabilization)
    while((SYSCTL_RIS_R&0x00000040)==0){};  // wait for PLLRIS bit
    // 6) when it is stablized, enable use of PLL by clearing BYPASS
    SYSCTL_RCC2_R &= ~0x00000800;
}
```

Example

- How would you change the program if your microcontroller had an 8 MHz crystal and you wish to run at 50 MHz?

Change the specification from 16 MHz to 8 MHz.
Change the line

```
SYSCTL_RCC_R += 0x00000540; // 10101, configure for 16 MHz crystal  0x15
```

to

```
SYSCTL_RCC_R += 0x00000380; // 01110, configure for 8 MHz crystal  0xE
```

Change the specification from divide by 5 to divide by 8.
Change the line

```
SYSCTL_RCC2_R += (4<<22); // configure for 80 MHz clock
```

to

```
SYSCTL_RCC2_R += (7<<22); // configure for 50 MHz clock
```

Timers (timing functions) in computer systems

- Periodically interrupt CPU to perform tasks
  - Sample sensor readings (temperature, pressure, etc.)
  - Generate music samples
- Provide accurate time delays
  - Instead of software loops
- Generate pulses or periodic waveforms
  - PWM signal for motor control
  - Strobe pulse for an external device
- Measure duration of an external event
  - Tachometer (revolutions per minute measuring device) signal period to measure motor speed

Timers on TM4C123GH6PM

1. General Purpose Timer (pp.704)
   - Many modes such as One-Shot/Periodic/Real time clock/Input Edge count/Input edge time/pwm mode
2. PWM Timer (pp.1230)
3. **24 bit system timer (SysTick) –standard in all Cortex-M CPUs (pp.123)**

General Purpose Timers

Table 11-3. General-Purpose Timer Capabilities

Mode	Timer Use	Count Direction	Counter Size		Prescaler Size[a]		Prescaler Behavior (Count Direction)
			16/32-bit GPTM	32/64-bit Wide GPTM	16/32-bit GPTM	32/64-bit Wide GPTM	
One-shot	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	N/A
Periodic	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Up), Prescaler (Down)
	Concatenated	Up or Down	32-bit	64-bit	-	-	N/A
RTC	Concatenated	Up	32-bit	64-bit	-	-	N/A
Edge Count	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
Edge Time	Individual	Up or Down	16-bit	32-bit	8-bit	16-bit	Timer Extension (Both)
PWM	Individual	Down	16-bit	32-bit	8-bit	16-bit	Timer Extension

a. The prescaler is only available when the timers are used individually

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
T1CCP0	30 58	PF2 (7) PB4 (7)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 0.
T1CCP1	31 57	PF3 (7) PB5 (7)	I/O	TTL	16/32-Bit Timer 1 Capture/Compare/PWM 1.
T2CCP0	5 45	PF4 (7) PB0 (7)	I/O	TTL	16/32-Bit Timer 2 Capture/Compare/PWM 0.
T2CCP1	46	PB1 (7)	I/O	TTL	16/32-Bit Timer 2 Capture/Compare/PWM 1.
T3CCP0	47	PB2 (7)	I/O	TTL	16/32-Bit Timer 3 Capture/Compare/PWM 0.
T3CCP1	48	PB3 (7)	I/O	TTL	16/32-Bit Timer 3 Capture/Compare/PWM 1.
T4CCP0	52	PC0 (7)	I/O	TTL	16/32-Bit Timer 4 Capture/Compare/PWM 0.
T4CCP1	51	PC1 (7)	I/O	TTL	16/32-Bit Timer 4 Capture/Compare/PWM 1.
T5CCP0	50	PC2 (7)	I/O	TTL	16/32-Bit Timer 5 Capture/Compare/PWM 0.
T5CCP1	49	PC3 (7)	I/O	TTL	16/32-Bit Timer 5 Capture/Compare/PWM 1.
WT0CCP0	16	PC4 (7)	I/O	TTL	32/64-Bit Wide Timer 0 Capture/Compare/PWM 0.
WT0CCP1	15	PC5 (7)	I/O	TTL	32/64-Bit Wide Timer 0 Capture/Compare/PWM 1.
WT1CCP0	14	PC6 (7)	I/O	TTL	32/64-Bit Wide Timer 1 Capture/Compare/PWM 0.
WT1CCP1	13	PC7 (7)	I/O	TTL	32/64-Bit Wide Timer 1 Capture/Compare/PWM 1.
WT2CCP0	61	PD0 (7)	I/O	TTL	32/64-Bit Wide Timer 2 Capture/Compare/PWM 0.
WT2CCP1	62	PD1 (7)	I/O	TTL	32/64-Bit Wide Timer 2 Capture/Compare/PWM 1.
WT3CCP0	63	PD2 (7)	I/O	TTL	32/64-Bit Wide Timer 3 Capture/Compare/PWM 0.
WT3CCP1	64	PD3 (7)	I/O	TTL	32/64-Bit Wide Timer 3 Capture/Compare/PWM 1.
WT4CCP0	43	PD4 (7)	I/O	TTL	32/64-Bit Wide Timer 4 Capture/Compare/PWM 0.
WT4CCP1	44	PD5 (7)	I/O	TTL	32/64-Bit Wide Timer 4 Capture/Compare/PWM 1.
WT5CCP0	53	PD6 (7)	I/O	TTL	32/64-Bit Wide Timer 5 Capture/Compare/PWM 0.
WT5CCP1	10	PD7 (7)	I/O	TTL	32/64-Bit Wide Timer 5 Capture/Compare/PWM 1.

a. The TTL designation indicates the pin has TTL-compatible voltage levels.

est_29072018_KeyPad\keypadAlif\keypadAlif\keypadA.uvprojx - µVision

Tools SVCS Window Help

Timer

main.c | keypad.c | system_TM4C123.c | startup_TM4C123.s | TM4C123GH6PM.h

```
511
512 typedef struct {                                    /*!< TIMER0 Structure
513   __IO uint32_t  CFG;                               /*!< GPTM Configuration
514   __IO uint32_t  TAMR;                              /*!< GPTM Timer A Mode
515   __IO uint32_t  TBMR;                              /*!< GPTM Timer B Mode
516   __IO uint32_t  CTL;                               /*!< GPTM Control
517   __IO uint32_t  SYNC;                              /*!< GPTM Synchronize
518   __I  uint32_t  RESERVED;
519   __IO uint32_t  IMR;                               /*!< GPTM Interrupt Mask
520   __IO uint32_t  RIS;                               /*!< GPTM Raw Interrupt Status
521   __IO uint32_t  MIS;                               /*!< GPTM Masked Interrupt Status
522   __O  uint32_t  ICR;                               /*!< GPTM Interrupt Clear
523   __IO uint32_t  TAILR;                             /*!< GPTM Timer A Interval Load
524   __IO uint32_t  TBILR;                             /*!< GPTM Timer B Interval Load
525   __IO uint32_t  TAMATCHR;                          /*!< GPTM Timer A Match
526   __IO uint32_t  TBMATCHR;                          /*!< GPTM Timer B Match
527   __IO uint32_t  TAPR;                              /*!< GPTM Timer A Prescale
528   __IO uint32_t  TBPR;                              /*!< GPTM Timer B Prescale
529   __IO uint32_t  TAPMR;                             /*!< GPTM TimerA Prescale Match
530   __IO uint32_t  TBPMR;                             /*!< GPTM TimerB Prescale Match
531   __IO uint32_t  TAR;                               /*!< GPTM Timer A
532   __IO uint32_t  TBR;                               /*!< GPTM Timer B
533   __IO uint32_t  TAV;                               /*!< GPTM Timer A Value
534   __IO uint32_t  TBV;                               /*!< GPTM Timer B Value
535   __IO uint32_t  RTCPD;                             /*!< GPTM RTC Predivide
536   __IO uint32_t  TAPS;                              /*!< GPTM Timer A Prescale Snapshot
537   __IO uint32_t  TBPS;                              /*!< GPTM Timer B Prescale Snapshot
538   __IO uint32_t  TAPV;                              /*!< GPTM Timer A Prescale Value
539   __IO uint32_t  TBPV;                              /*!< GPTM Timer B Prescale Value
540   __I  uint32_t  RESERVED1[981];
541   __IO uint32_t  PP;                                /*!< GPTM Peripheral Properties
542 } TIMER0_Type;
```

```
void timer0Init(unsigned long Hertz, BOOL irqEn ){
        TIMER0_Type * Timer;
        Timer = TIMER0;
        SYSCTL->RCGCTIMER |= 0x01;
/*1. Ensure the timer is disabled, TnEN GPTMCTL */
        Timer->CTL &= ~ 0x01;
/*2. write GPTM cofiguration register (GPTMCFG) with 0x00000000 */
        Timer->CFG = 0x00000000;
/*3. Configue the TnMr field in the GPTM timer n Mode Register GOTMTnMR */
/*3a or b 0x1 for One-Shot mode or 0x02 for Periodic mode */
        Timer->TAMR |= 0x2;
/*4. Optionally...*/
        Timer->TAMR |= (1<<4); //down counter
/*5. Load the start value into the GPTM Timer n Interval load register (GPTMTnILR) */
        Timer->TAILR = (SystemClock / Hertz)-1; //count value
/*6. If interrupts are required, set the bits in the GPTM interrupt mark register GPTMIMR */
        if(irqEn) Timer->IMR |= 0x01;
        else Timer->IMR &= ~0x01;
/*7. set TnEN bit in the GPTMCTL register to enable the timer and start counting */
/*8. Poll the GPTMRIS register or wait for the interrupt to generated(if enabled). In
   both cases, the status flags are cleared by writing a 1 to the appropriate bit
  of the GPTM Interrupt clear Register GPTMICR*/
        if(irqEn)               __NVIC_EnableIRQ(TIMER0A_IRQn); //irqEn →Boolean type
        Timer->CTL |= 0x01;
}
```

```
void TIMER0A_Handler(void){
        TIMER0->ICR |= 0x01; //ack.
        //Do something
        }
```

Register 10: GPTM Timer A Interval Load (GPTMTAILR), offset 0x028

When the timer is counting down, this register is used to load the starting count value into the timer. When the timer is counting up, this register sets the upper bound for the timeout event.

When a 16/32-bit GPTM is configured to one of the 32-bit modes, **GPTMTAILR** appears as a 32-bit register (the upper 16-bits correspond to the contents of the **GPTM Timer B Interval Load (GPTMTBILR)** register). In a 16-bit mode, the upper 16 bits of this register read as 0s and have no effect on the state of **GPTMTBILR**.

When a 32/64-bit Wide GPTM is configured to one of the 64-bit modes, **GPTMTAILR** contains bits 31:0 of the 64-bit count and the **GPTM Timer B Interval Load (GPTMTBILR)** register contains bits 63:32.

GPTM Timer A Interval Load (GPTMTAILR)

16/32-bit Timer 0 base: 0x4003.0000
16/32-bit Timer 1 base: 0x4003.1000
16/32-bit Timer 2 base: 0x4003.2000
16/32-bit Timer 3 base: 0x4003.3000
16/32-bit Timer 4 base: 0x4003.4000
16/32-bit Timer 5 base: 0x4003.5000
32/64-bit Wide Timer 0 base: 0x4003.6000
32/64-bit Wide Timer 1 base: 0x4003.7000
32/64-bit Wide Timer 2 base: 0x4004.C000
32/64-bit Wide Timer 3 base: 0x4004.D000
32/64-bit Wide Timer 4 base: 0x4004.E000
32/64-bit Wide Timer 5 base: 0x4004.F000
Offset 0x028
Type RW, reset 0xFFFF.FFFF

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TAILR															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TAILR															
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Bit/Field	Name	Type	Reset	Description
31:0	TAILR	RW	0xFFFF.FFFF	GPTM Timer A Interval Load Register Writing this field loads the counter for Timer A. A read returns the current value of **GPTMTAILR**.

Periodic basis interrupts

- A computer to request an interrupt on a periodic basis
- An interrupt handler will be executed at fixed time intervals.
- Essential for the implementation of real-time data acquisition and real-time control systems.
  - Because software servicing must be performed at accurate time intervals.

For example

Implementing a digital controller that executes a control algorithm 100 times per a second

- Set up the internal timer hardware to request an interrupt every 10ms
- The ISR will execute the digital control algorithm and then return to the main thread periodically

For example

Perform analog input and/or analog output: sample the ADC 100 times a second

- Set up the internal timer hardware to request an interrupt every 10 ms.
- The ISR will sample the ADC, process (or save) the data, and then return to the main thread periodically

PWM Timers

Pin Name	Pin Number	Pin Mux / Pin Assignment	Pin Type	Buffer Type[a]	Description
M0FAULT0	30 53 63	PF2 (4) PD6 (4) PD2 (4)	I	TTL	Motion Control Module 0 PWM Fault 0.
M0PWM0	1	PB6 (4)	O	TTL	Motion Control Module 0 PWM 0. This signal is controlled by Module 0 PWM Generator 0.
M0PWM1	4	PB7 (4)	O	TTL	Motion Control Module 0 PWM 1. This signal is controlled by Module 0 PWM Generator 0.
M0PWM2	58	PB4 (4)	O	TTL	Motion Control Module 0 PWM 2. This signal is controlled by Module 0 PWM Generator 1.
M0PWM3	57	PB5 (4)	O	TTL	Motion Control Module 0 PWM 3. This signal is controlled by Module 0 PWM Generator 1.
M0PWM4	59	PE4 (4)	O	TTL	Motion Control Module 0 PWM 4. This signal is controlled by Module 0 PWM Generator 2.
M0PWM5	60	PE5 (4)	O	TTL	Motion Control Module 0 PWM 5. This signal is controlled by Module 0 PWM Generator 2.
M0PWM6	16 61	PC4 (4) PD0 (4)	O	TTL	Motion Control Module 0 PWM 6. This signal is controlled by Module 0 PWM Generator 3.
M0PWM7	15 62	PC5 (4) PD1 (4)	O	TTL	Motion Control Module 0 PWM 7. This signal is controlled by Module 0 PWM Generator 3.
M1FAULT0	5	PF4 (5)	I	TTL	Motion Control Module 1 PWM Fault 0.
M1PWM0	61	PD0 (5)	O	TTL	Motion Control Module 1 PWM 0. This signal is controlled by Module 1 PWM Generator 0.
M1PWM1	62	PD1 (5)	O	TTL	Motion Control Module 1 PWM 1. This signal is controlled by Module 1 PWM Generator 0.
M1PWM2	23 59	PA6 (5) PE4 (5)	O	TTL	Motion Control Module 1 PWM 2. This signal is controlled by Module 1 PWM Generator 1.
M1PWM3	24 60	PA7 (5) PE5 (5)	O	TTL	Motion Control Module 1 PWM 3. This signal is controlled by Module 1 PWM Generator 1.
M1PWM4	28	PF0 (5)	O	TTL	Motion Control Module 1 PWM 4. This signal is controlled by Module 1 PWM Generator 2.
M1PWM5	29	PF1 (5)	O	TTL	Motion Control Module 1 PWM 5. This signal is controlled by Module 1 PWM Generator 2.
M1PWM6	30	PF2 (5)	O	TTL	Motion Control Module 1 PWM 6. This signal is controlled by Module 1 PWM Generator 3.
M1PWM7	31	PF3 (5)	O	TTL	Motion Control Module 1 PWM 7. This signal is controlled by Module 1 PWM Generator 3.

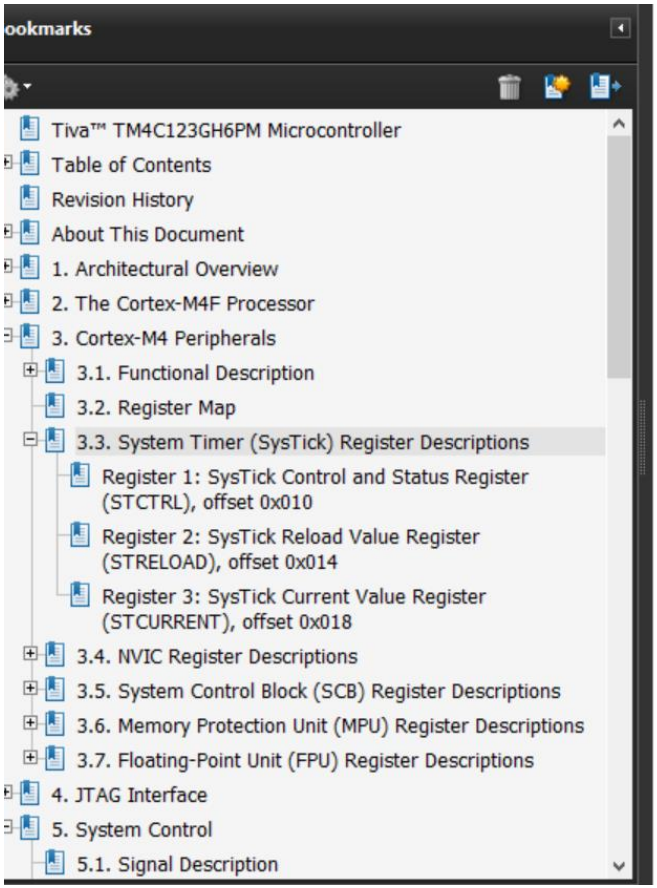
a. The TTL designation indicates the pin has TTL-compatible voltage levels.

More elaborated, controllable version of PWM

SysTick

- System tick timers
- A 24-bit down counter driven by a system clock
  - We can choose the clock type (internal, external..)
- Initiating an action on a periodic basis.

Pp 123



3.1.1 System Timer (SysTick)

Cortex-M4 includes an integrated system timer, SysTick, which provides a simple, 24-bit clear-on-write, decrementing, wrap-on-zero counter with a flexible control mechanism. The counter can be used in several different ways, for example as:

- An RTOS tick timer that fires at a programmable rate (for example, 100 Hz) and invokes a SysTick routine.
- A high-speed alarm timer using the system clock.
- A variable rate alarm or signal timer—the duration is range-dependent on the reference clock used and the dynamic range of the counter.
- A simple counter used to measure time to completion and time used.
- An internal clock source control based on missing/meeting durations. The COUNT bit in the **STCTRL** control and status register can be used to determine if an action completed within a set duration, as part of a dynamic clock management control loop.

The timer consists of three registers:

- **SysTick Control and Status (STCTRL)**: A control and status counter to configure its clock, enable the counter, enable the SysTick interrupt, and determine counter status.
- **SysTick Reload Value (STRELOAD)**: The reload value for the counter, used to provide the counter's wrap value.
- **SysTick Current Value (STCURRENT)**: The current value of the counter.

When enabled, the timer counts down on each clock from the reload value to zero, reloads (wraps) to the value in the **STRELOAD** register on the next clock edge, then decrements on subsequent clocks. Clearing the STRELOAD register disables the counter on the next wrap. When the counter

- Timer/Counter operation
  - 24-bit counter *decrements* (down counter) at bus clock frequency
    - With 12 MHz bus clock, decrements every 83.3 ns
    - With 50 MHz bus clock, decrements every 20 ns
    - With 80 MHz bus clock, decrements every 12.5 ns

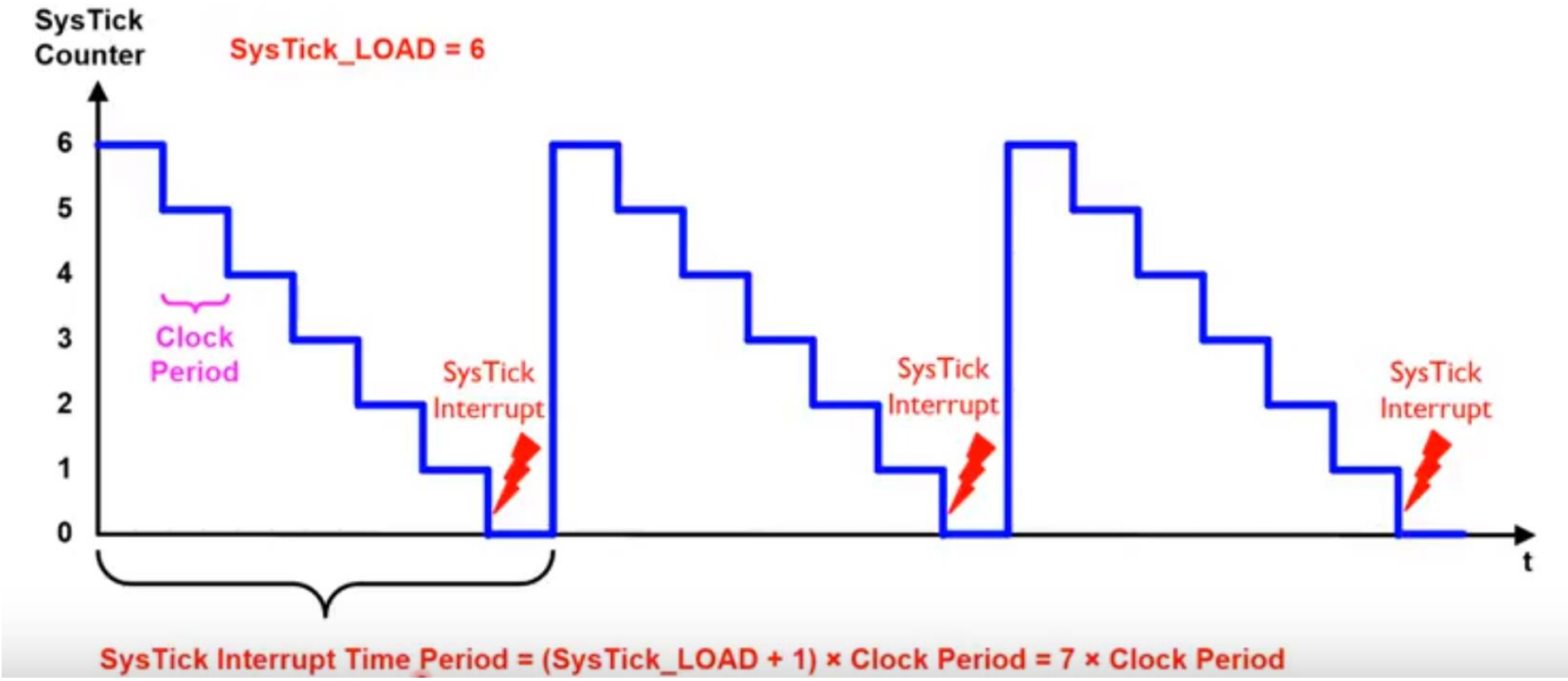
– Counting is from n *(reload value)* → 0

- Setting *n* appropriately will make the counter a modulo *n*+1 counter. That is:
  - `next_value = (current_value-1) mod (n+1)`
  - When a < n, (a mod n) is simply a

Sequence:
n,n-1,n-2,n-3… 2,1,0,n,n-1…

Set a flag

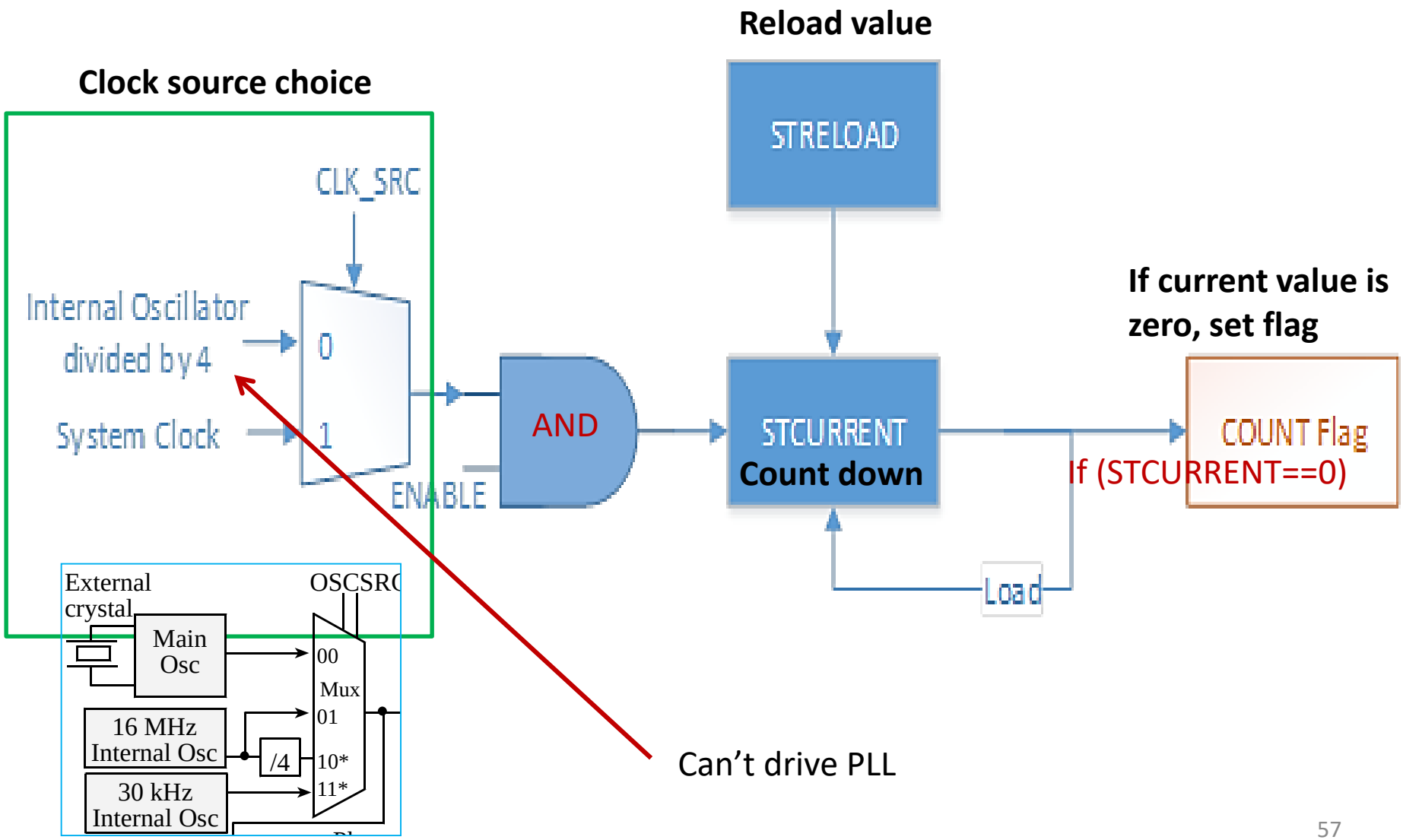
Set with Reload value n


SysTick Counter
SysTick_LOAD = 6
6
5
4
3
2
1
0
Clock Period
SysTick Interrupt
SysTick Interrupt
SysTick Interrupt
t
SysTick Interrupt Time Period = (SysTick_LOAD + 1) × Clock Period = 7 × Clock Period

Example:
SysTick to read a sensor every 100ms

1. Setting up a system clock using PLL (optional)
2. Counting down from an initial value to 0
   - Every cycle, it decreases 1
3. When it reaches 0
   - in the next clock, it underflows
   - and raises a flag called COUNT
   - reloads the initial value
4. Repeating over and again

SysTick structure diagram



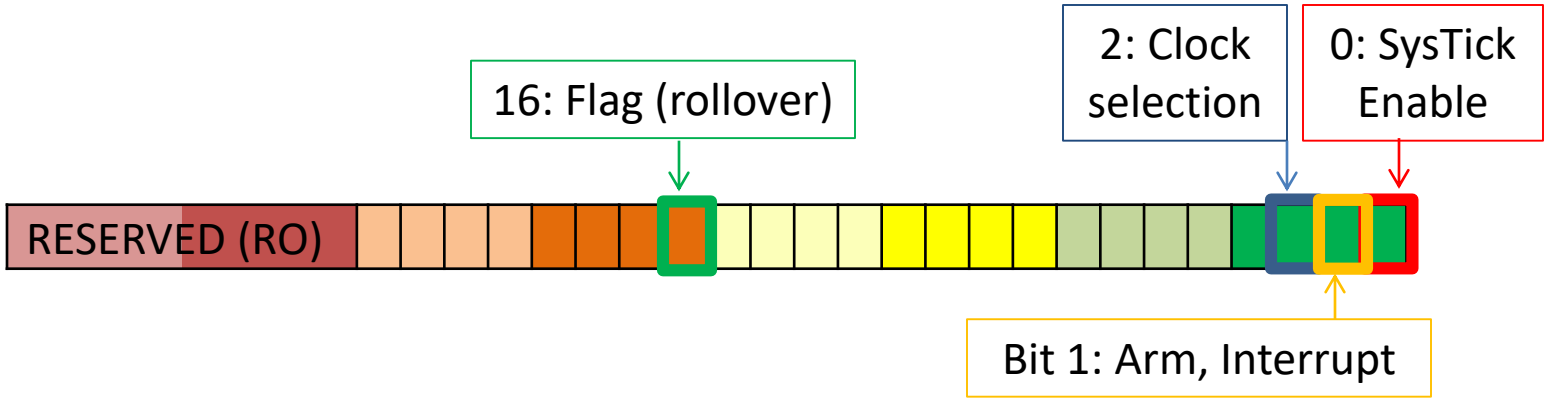
3 Registers for SysTick

1. Control
2. SysTick Current Value Register
3. SysTick Reload Value Register

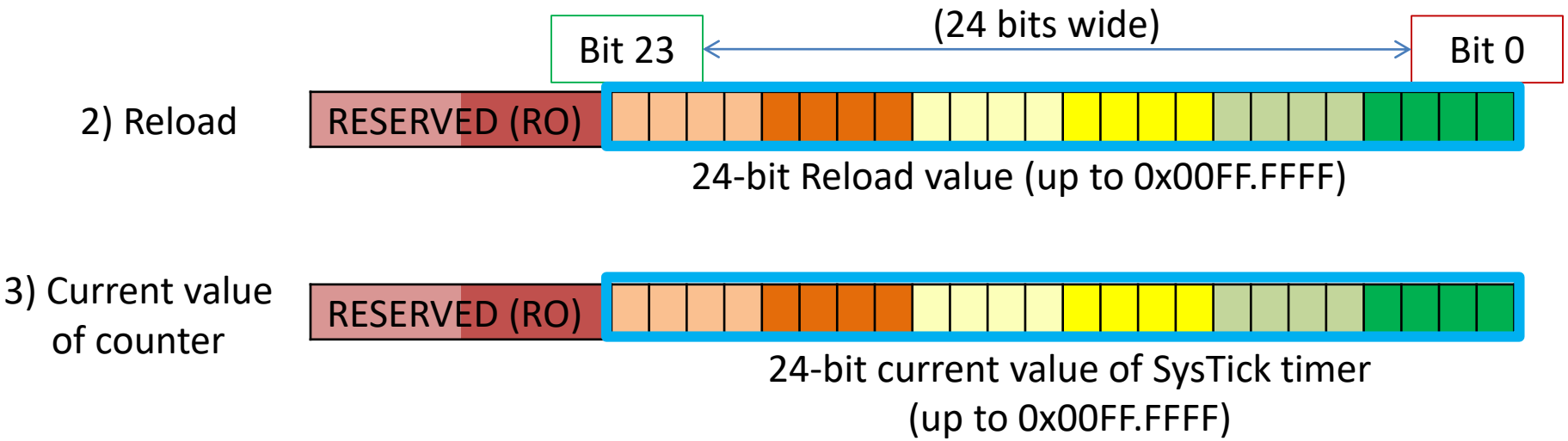
Address	31-24	23-17	16	15-3	2	1	0	Name
$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

We don’t use: 24 bit counter

1) CTRL (control)

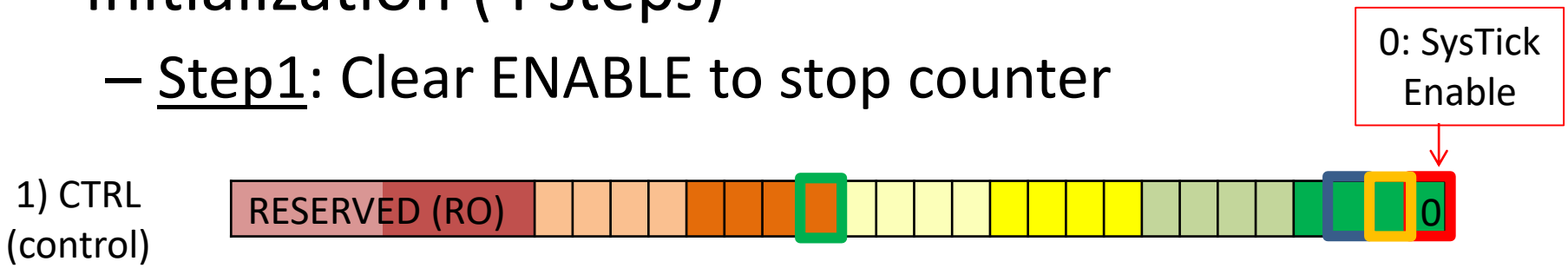


Bit	Description
0: Enable	0: the counter is disabled; 1: enables SysTick to begin counting down
1: INTEN	0: interrupt generation is disabled; 1: when SysTick counts to 0 an interrupt is generated
2: CLK_SRC	0: internal oscillator (16 MHz) divided by 4; (no PLL) 1: System Clock
16: Flag	0: the SysTick has not counted down to zero yet 1: the SysTick has counted down to zero Note: this flag is cleared by reading the STCTL register or writing to STCURRENT register


(24 bits wide)
Bit 23
Bit 0
2) Reload
RESERVED (RO)
24-bit Reload value (up to 0x00FF.FFFF)
3) Current value of counter
RESERVED (RO)
24-bit current value of SysTick timer (up to 0x00FF.FFFF)

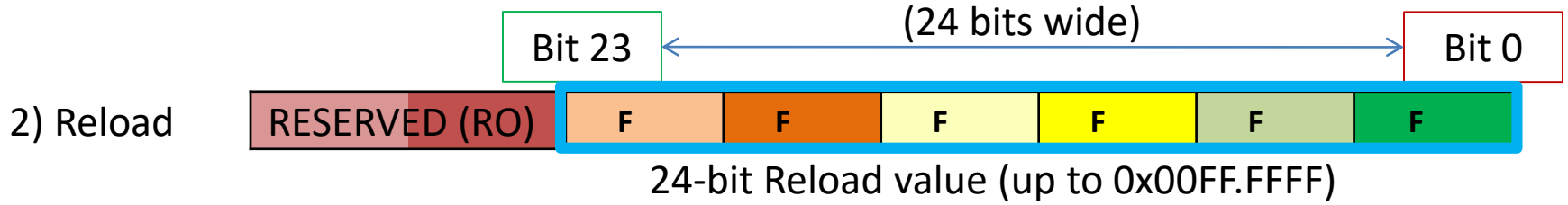
SysTick Init

- Initialization (4 steps)
  - Step1: Clear ENABLE to stop counter




```
i.e., NVIC_ST_CTRL_R = 0;
```

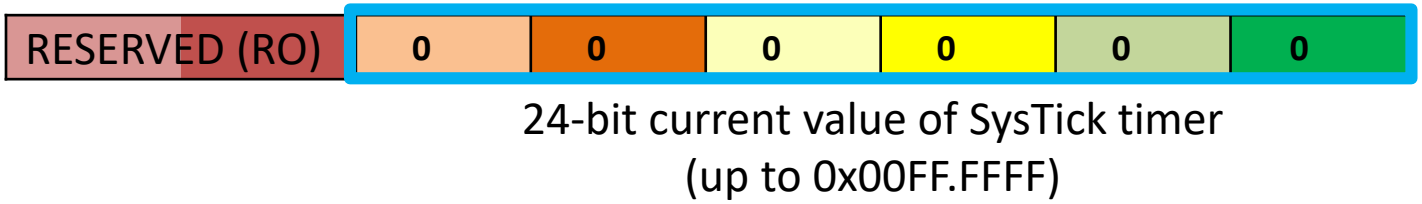
  - Step2: Specify the RELOAD value




```
i.e., NVIC_ST_RELOAD_R = 0x00FFFFFF
```

– Step3: Clear the counter via NVIC_ST_CURRENT_R

3) Current value of counter

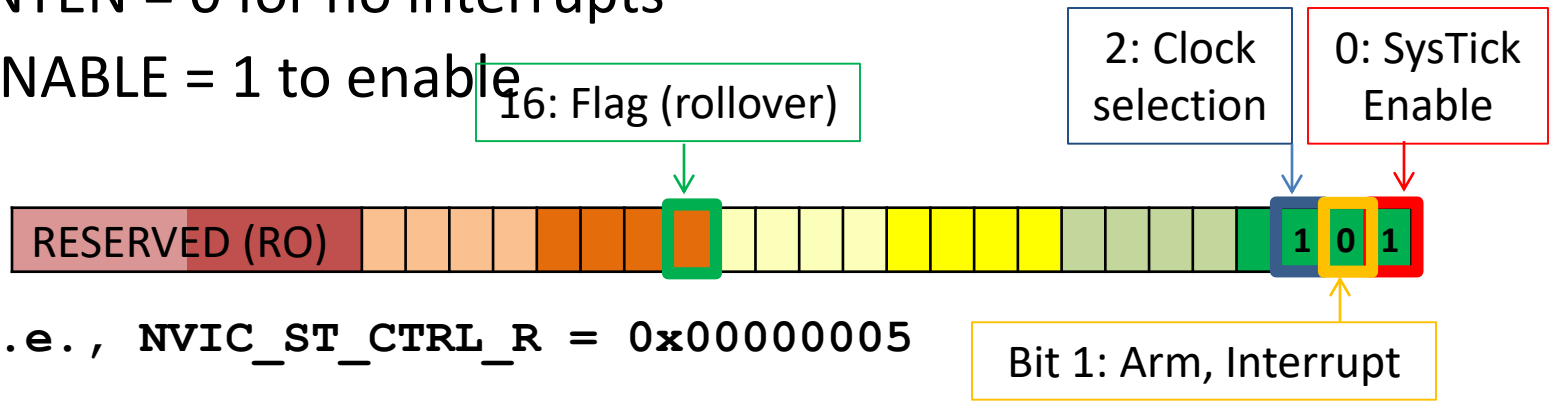



```
i.e., NVIC_ST_CURRENT_R = 0;
```

– Step4: Set NVIC_ST_CTRL_R

- CLK_SRC = 1 (system clk. If PLL activate, we will use PLL clock or 0:internal clk./4)
- INTEN = 0 for no interrupts
- ENABLE = 1 to enable

1) CTRL (control)

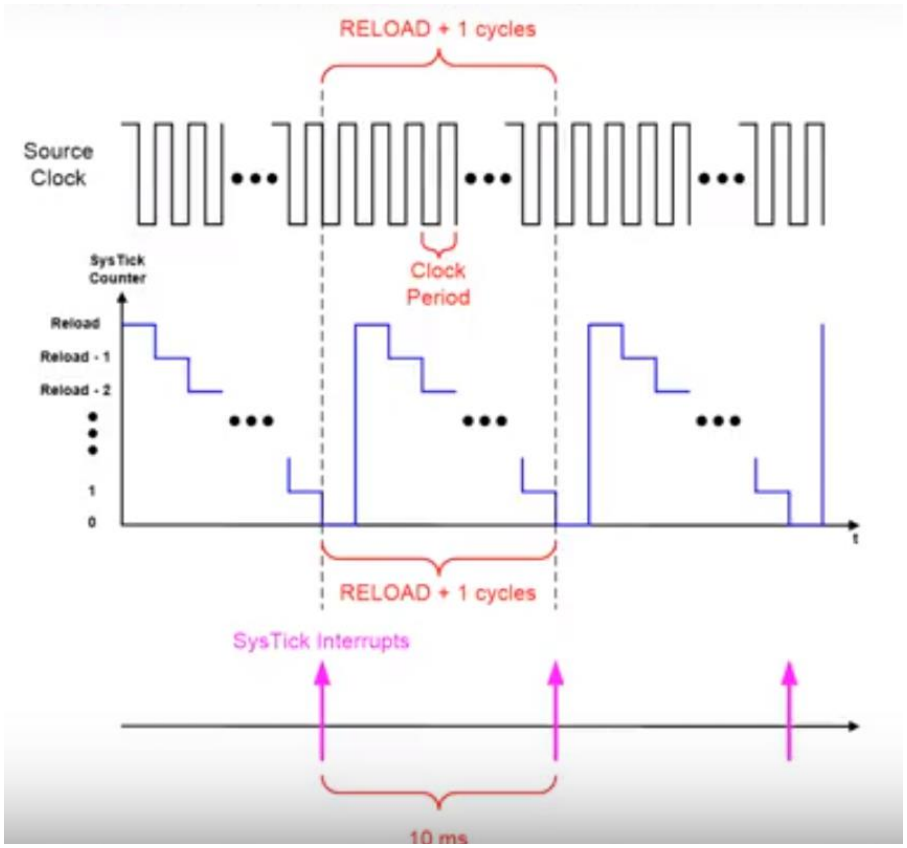



```
i.e., NVIC_ST_CTRL_R = 0x00000005
```

Suppose clock source = 80MHz

Goal: SysTick Interval = 10ms

What is RELOAD value?



$$Reload = \frac{10\ ms}{Clock\ Period} - 1$$

$$= 10ms \times Clock\ Frequency\ - 1$$

$$= 10ms \times 80MHz - 1$$

$$= 10 \times 10^{-3} \times 80 \times 10^{6} - 1$$

$$= 800000 - 1$$

$$= 799999$$

SysTick Timer Init examples:

```
#define NVIC_ST_CTRL_R(*((volatile uint32_t *)0xE000E010))
#define NVIC_ST_RELOAD_R(*((volatile uint32_t *)0xE000E014))
#define NVIC_ST_CURRENT_R(*((volatile uint32_t *)0xE000E018))
```

- General case

```
void SysTick_Init(void){
 NVIC_ST_CTRL_R = 0; // 1) disable SysTick during setup
 NVIC_ST_RELOAD_R = 0x00FFFFFF; // 2) maximum reload value
 NVIC_ST_CURRENT_R = 0; // 3) any write to CURRENT clears it
 NVIC_ST_CTRL_R = 0x05; // 4) enable SysTick with core clock 0101
}
```

- Interrupt available

```
void SysTick_Init(uint32_t period){
  Counts = 0;
  NVIC_ST_CTRL_R = 0;              // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;     // reload value
  NVIC_ST_CURRENT_R = 0;           // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
  NVIC_ST_CTRL_R = 0x07;           // enable, core bus clock, arm 0111
}
```

SysTick periodic interrupt

- A simple way to create periodic interrupts
- A periodic interrupt is one that is requested on a fixed time basis.
- SysTick is a 24-bit counter that decrements at the bus clock frequency.

107p

Figure 2-6. Vector Table

Exception number	IRQ number	Offset	Vector
154	138	0x0268	IRQ131
.		.	.
.		.	.
.		.	.
		0x004C	
18	2	0x0048	IRQ2
17	1	0x0044	IRQ1
16	0	0x0040	IRQ0
15	-1	0x003C	Systick
14	-2	0x0038	PendSV
13			Reserved
12			Reserved for Debug
11	-5	0x002C	SVCall
10			Reserved
9			
8			
7			
6	-10	0x0018	Usage fault
5	-11	0x0014	Bus fault
4	-12	0x0010	Memory management fault
3	-13	0x000C	Hard fault
2	-14	0x0008	NMI
1		0x0004	Reset
		0x0000	Initial SP value

INTERRUPT VECTORS

Vector address	Exception Number	IRQ	ISR name in Startup.s	NVIC	Priority bits
0x00000038	14	-2	PendSV_Handler	NVIC_SYS_PRI3_R	23 – 21
0x0000003C	15	-1	SysTick_Handler	NVIC_SYS_PRI3_R	31 – 29
0x00000040	16	0	GPIOPortA_Handler	NVIC_PRI0_R	7 – 5
0x00000044	17	1	GPIOPortB_Handler	NVIC_PRI0_R	15 – 13
0x00000048	18	2	GPIOPortC_Handler	NVIC_PRI0_R	23 – 21
0x0000004C	19	3	GPIOPortD_Handler	NVIC_PRI0_R	31 – 29
0x00000050	20	4	GPIOPortE_Handler		
0x00000054	21	5	UART0_Handler		
0x00000058	22	6	UART1_Handler		
0x0000005C	23	7	SSI0_Handler		
0x00000060	24	8	I2C0_Handler		
0x00000064	25	9	PWMFault_Handler		
0x00000068	26	10	PWM0_Handler		
0x0000006C	27	11	PWM1_Handler		
0x00000070	28	12	PWM2_Handler		
0x00000074	29	13	Quadrature0_Handler	NVIC_PRI3_R	15 – 13
0x00000078	30			NVIC_PRI3_R	23 – 21
0x0000007C	31			NVIC_PRI3_R	31 – 29
0x00000080	32			NVIC_PRI4_R	7 – 5
0x00000084	33			NVIC_PRI4_R	15 – 13
0x00000088	34			NVIC_PRI4_R	23 – 21
0x0000008C	35	19	Timer0A_Handler	NVIC_PRI4_R	31 – 29
0x00000090	36	20	Timer0B_Handler	NVIC_PRI5_R	7 – 5
0x00000094	37	21	Timer1A_Handler	NVIC_PRI5_R	15 – 13
0x00000098	38	22	Timer1B_Handler	NVIC_PRI5_R	23 – 21
0x0000009C	39	23	Timer2A_Handler	NVIC_PRI5_R	31 – 29
0x000000A0	40	24	Timer2B_Handler	NVIC_PRI6_R	7 – 5
0x000000A4	41	25	Comp0_Handler	NVIC_PRI6_R	15 – 13
0x000000A8	42	26	Comp1_Handler	NVIC_PRI6_R	23 – 21
0x000000AC	43	27	Comp2_Handler	NVIC_PRI6_R	31 – 29
0x000000B0	44	28	SysCtl_Handler	NVIC_PRI7_R	7 – 5
0x000000B4	45	29	FlashCtl_Handler	NVIC_PRI7_R	15 – 13
0x000000B8	46	30	GPIOPortF_Handler	NVIC_PRI7_R	23 – 21
0x000000BC	47	31	GPIOPortG_Handler	NVIC_PRI7_R	31 – 29
0x000000C0	48	32	GPIOPortH_Handler	NVIC_PRI8_R	7 – 5
0x000000C4	49	33	UART2_Handler	NVIC_PRI8_R	15 – 13
0x000000C8	50	34	SSI1_Handler	NVIC_PRI8_R	23 – 21
0x000000CC	51	35	Timer3A_Handler	NVIC_PRI8_R	31 – 29
0x000000D0	52	36	Timer3B_Handler	NVIC_PRI9_R	7 – 5
0x000000D4	53	37	I2C1_Handler	NVIC_PRI9_R	15 – 13
0x000000D8	54	38	Quadrature1_Handler	NVIC_PRI9_R	23 – 21
0x000000DC	55	39	CAN0_Handler	NVIC_PRI9_R	31 – 29
0x000000E0	56	40	CAN1_Handler	NVIC_PRI10_R	7 – 5
0x000000E4	57	41	CAN2_Handler	NVIC_PRI10_R	15 – 13
0x000000E8	58	42	Ethernet_Handler	NVIC_PRI10_R	23 – 21
0x000000EC	59	43	Hibernate_Handler	NVIC_PRI10_R	31 – 29
0x000000F0	60	44	USB0_Handler	NVIC_PRI11_R	7 – 5
0x000000F4	61	45	PWM3_Handler	NVIC_PRI11_R	15 – 13
0x000000F8	62	46	uDMA_Handler	NVIC_PRI11_R	23 – 21
0x000000FC	63	47	uDMA_Error	NVIC_PRI11_R	31 – 29

Systick interrupt handler and address

(Systick interrupt priority control)
Nested vector interrupt controller and specific bits to control (different from handler)

The SysTick registers

- Used to create a periodic interrupt

Address	31-24	23-17	16	15-3	2	1	0	Name
0xE000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	**NVIC_ST_CTRL_R**
0xE000E014	0	24-bit RELOAD value						**NVIC_ST_RELOAD_R**
0xE000E018	0	24-bit CURRENT value of SysTick counter						**NVIC_ST_CURRENT_R**

Address	31-29	28-24	23-21	20-8	7-5	4-0	Name
0xE000ED20	TICK	0	PENDSV	0	DEBUG	0	**NVIC_SYS_PRI3_R**

NVIC Registers

- **High order three bits of each byte define priority**

Address	31 – 29	23 – 21	15 – 13	7 – 5	Name
0xE000E400	GPIO Port D	GPIO Port C	GPIO Port B	GPIO Port A	NVIC_PRI0_R
0xE000E404	SSI0, Rx Tx	UART1, Rx Tx	UART0, Rx Tx	GPIO Port E	NVIC_PRI1_R
0xE000E408	PWM Gen 1	PWM Gen 0	PWM Fault	I2C0	NVIC_PRI2_R
0xE000E40C	ADC Seq 1	ADC Seq 0	Quad Encoder	PWM Gen 2	NVIC_PRI3_R
0xE000E410	Timer 0A	Watchdog	ADC Seq 3	ADC Seq 2	NVIC_PRI4_R
0xE000E414	Timer 2A	Timer 1B	Timer 1A	Timer 0B	NVIC_PRI5_R
0xE000E418	Comp 2	Comp 1	Comp 0	Timer 2B	NVIC_PRI6_R
0xE000E41C	GPIO Port G	GPIO Port F	Flash Control	System Control	NVIC_PRI7_R
0xE000E420	Timer 3A	SSI1, Rx Tx	UART2, Rx Tx	GPIO Port H	NVIC_PRI8_R
0xE000E424	CAN0	Quad Encoder 1	I2C1	Timer 3B	NVIC_PRI9_R
0xE000E428	Hibernate	Ethernet	CAN2	CAN1	NVIC_PRI10_R
0xE000E42C	uDMA Error	uDMA Soft Tfr	PWM Gen 3	USB0	NVIC_PRI11_R
0xE000ED20	**SysTick**	PendSV	--	Debug	NVIC_SYS_PRI3_R

SysTick Interrupt procedure

- f_{BUS} : the frequency of the bus clock
- n : the value of the **RELOAD** register
- The frequency of the periodic interrupt : $f_{BUS}/(n+1)$.

1) Clear the **ENABLE** bit to turn off SysTick during initialization.
2) Set the **RELOAD** register with the value n.
3) Write any value to **NVIC_ST_CURRENT_R** to clear the counter.

4) Write the desired mode to the control register, **NVIC_ST_CTRL_R**.
   - We must set **CLK_SRC**=1, because **CLK_SRC**=0 external clock mode is not implemented on the LM3S/TM4C family.
   - We need to set the **ENABLE**bit so the counter will run.
   - We set **INTEN** to enable interrupts.
5) We establish the priority of the SysTick interrupts using the TICK field in the **NVIC_SYS_PRI3_R** register.

An important notice

- SysTick is the only interrupt on the TM4C that has an automatic acknowledge.
- Notice there is no explicit software step in the ISR to clear the **COUNT** flag.

SysTick periodic interrupt

Systick_Periodic_Interrupt_Example.c*

```c
volatile unsigned long Counts=0; //Counts number of systick handler execution

void SysTick_Init(unsigned long period){
  NVIC_ST_CTRL_R = 0;          // disable SysTick during setup
  NVIC_ST_RELOAD_R = period-1;// reload value
  NVIC_ST_CURRENT_R = 0;       // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
  NVIC_ST_CTRL_R = 0x07; // enable SysTick with core clock and interrupts
                         // enable interrupts after all initialization is finished
}
void SysTick_Handler(void){
  GPIO_PORTF_DATA_R ^= 0x04;        // toggle PF2
  Counts = Counts + 1;
}
int main(void){ // running at 16 MHz
  SYSCTL_RCGC2_R |= 0x00000020; // activate port F
  Counts = 0;
  GPIO_PORTF_DIR_R |= 0x04;    // make PF2 output (PF2 built-in LED)
  GPIO_PORTF_AFSEL_R &= ~0x04;// disable alt funct on PF2
  GPIO_PORTF_DEN_R |= 0x04;    // enable digital I/O on PF2
  GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF0FF)+0x00000000;
  GPIO_PORTF_AMSEL_R = 0;      // disable analog functionality on PF
  SysTick_Init(16000000);      // initialize SysTick timer, every 1s, a fixed period
  EnableInterrupts();        // enable after everything initialized
  while(1){                    // interrupts every 1ms, 500 Hz flash
    WaitForInterrupt();
  }
}
```

PWM via SysTick

- Basic idea: a constant period, variable duty cycle

```
unsigned long H,L;
void Motor_Init(void){
  SYSCTL_RCGC2_R |= 0x00000001; // activate clock for port A
  H = L = 40000;              // 50%
  GPIO_PORTA_AMSEL_R &= ~0x20;     // disable analog functionality on PA5
  GPIO_PORTA_PCTL_R &= ~0x00F00000; // configure PA5 as GPIO
  GPIO_PORTA_DIR_R |= 0x20;    // make PA5 out
  GPIO_PORTA_DR8R_R |= 0x20;   // enable 8 mA drive on PA5
  GPIO_PORTA_AFSEL_R &= ~0x20;  // disable alt funct on PA5
  GPIO_PORTA_DEN_R |= 0x20;    // enable digital I/O on PA5
  GPIO_PORTA_DATA_R &= ~0x20;   // make PA5 low
  NVIC_ST_CTRL_R = 0;          // disable SysTick during setup
  NVIC_ST_RELOAD_R = L-1;      // reload value for 500us
  NVIC_ST_CURRENT_R = 0;       // any write to current clears it
  NVIC_SYS_PRI3_R = (NVIC_SYS_PRI3_R&0x00FFFFFF)|0x40000000; // priority 2
  NVIC_ST_CTRL_R = 0x00000007;  // enable with core clock and interrupts
}
void SysTick_Handler(void){
  if(GPIO_PORTA_DATA_R&0x20){   // check PA5
    GPIO_PORTA_DATA_R &= ~0x20; // make PA5 low
    NVIC_ST_RELOAD_R = L-1;     // reload value for low phase
  } else{
    GPIO_PORTA_DATA_R |= 0x20;  // make PA5 high
    NVIC_ST_RELOAD_R = H-1;     // reload value for high phase
  }
}
```

Reading	
Vol.1	**Vol.2**
Ch4 (4.3, 4.4)	Ch.2 (2.5, 2.6…)