Basic Timers

ECE 362
https://engineering.purdue.edu/ece362/

Reading Assignment

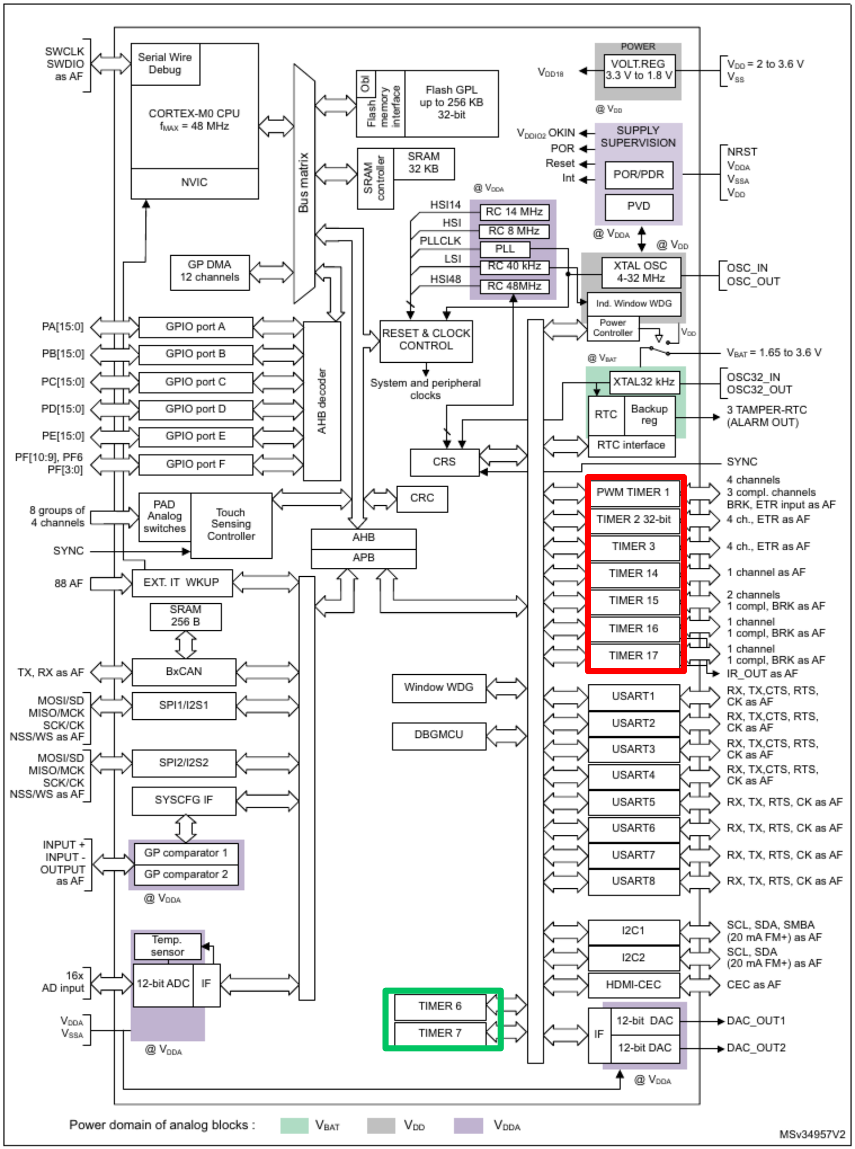
- Reading assignment:
  - Textbook, Chapter 15, “General-purpose Timers”, pages 373 – 414.
    - If you just read section 15.1 you will do yourself a tremendous favor.
    - Talking about it in this lecture.
  - Family Reference Manual, Chapter 20, "Basic timer (TIM6/TIM7)", pages 539 – 551.
  - Family Reference Manual, Chapter 17, "General purpose timers (TIM2 and TIM3)", pages 377 – 443.
  - Textbook, Chapter 10, “Mixing C and Assembly”, pages 215 – 236.
    - We’ll talk about this in the next lecture module.

STM32F091RCT6 has 9 timers

- TIM1
- TIM2 / TIM3
- TIM6 / TIM7 (the simplest ones)
- TIM14
- TIM15 / TIM16 / TIM17

- Why are TIM6/7 simplest?

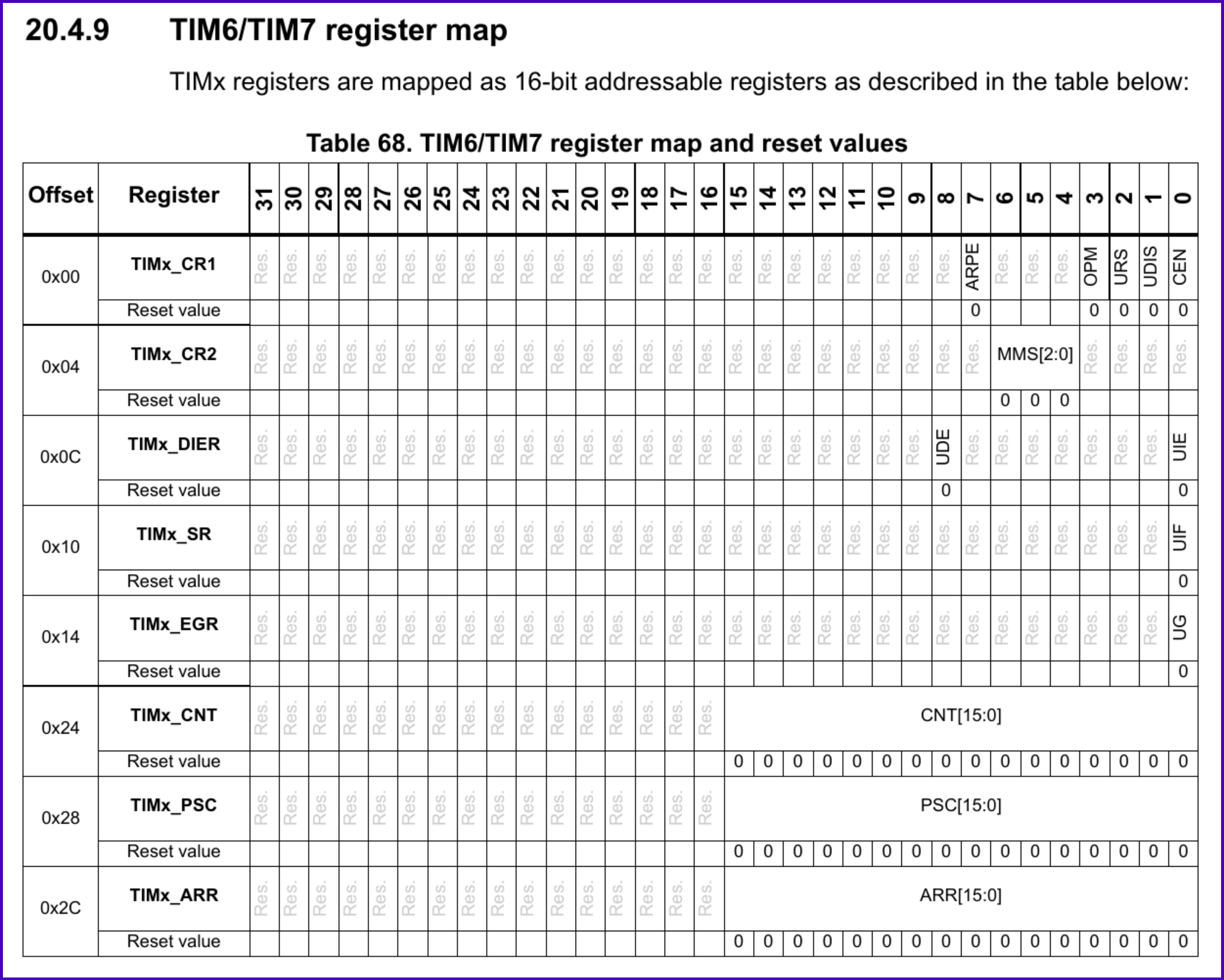
TIM6/7 have no external interfaces



- Others timers have special purposes as well as various external interfaces.
- TIM6/7 are just timers.
  - No external interfaces.
  - Only generate:
    - Interrupts
    - DMA events
    - DAC triggers

Don’t have to care about these two yet

TIM6/7 regs



20.4.9 TIM6/TIM7 register map

TIMx registers are mapped as 16-bit addressable registers as described in the table below:

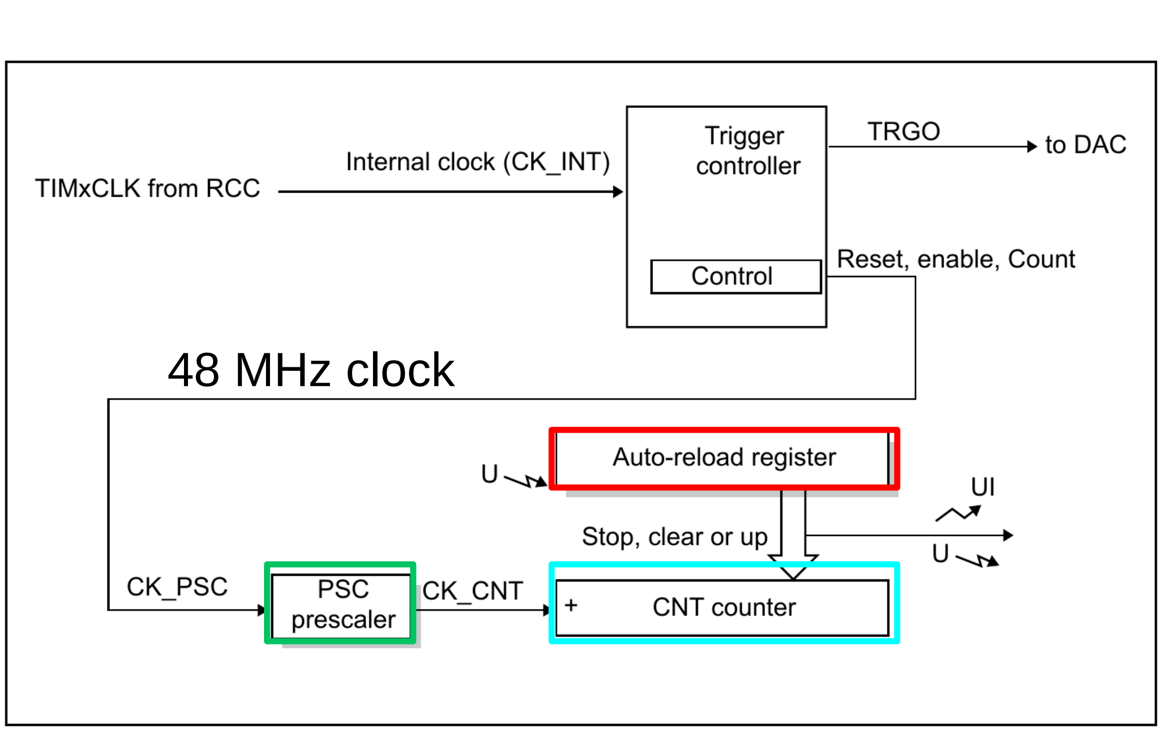
Table 68. TIM6/TIM7 register map and reset values

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	TIMx_CR1	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
	Reset value																									0				0	0	0	0
0x04	TIMx_CR2	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	MMS[2:0]			Res.	Res.	Res.	Res.
	Reset value																										0	0	0				
0x0C	TIMx_DIER	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UDE	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UIE
	Reset value																								0								0
0x10	TIMx_SR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UIF
	Reset value																																0
0x14	TIMx_EGR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UG
	Reset value																																0
0x24	TIMx_CNT	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CNT[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x28	TIMx_PSC	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PSC[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x2C	TIMx_ARR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ARR[15:0]															
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

There are a few things we’ll cover when we look at more complicated timers.

Organization

Figure 193. Basic timer block diagram



Three main 16-bit registers:

- PSC: Prescaler
  - Divides 48 MHz by 1–65536
- CNT: Free-running counter
  - Counts up from 0 to ARR value. Then updated to 0.
- ARR: Auto-reload register
  - Max count for CNT.

Basic timer function

- The prescaler divides the system clock to produce CK_CNT.
- Once enabled, the counter (CNT) counts up by one on every tick of the clock (CK_CNT).
- The counter counts up to the value of the ARR.
  - On the next CK_CNT tick:
    - The counter is reset to zero.
    - An update event occurs.

Important notes about PSC,ARR

- The prescaler and auto-reload register are N+1 values.
  - If you want to divide the 48 MHz clock by 48 to produce a 1 MHz clock, you write 47 to the PSC.
  - When PSC == 0, it means divide by 1 (effectively, no prescaler).
  - When ARR == 0, **it won’t do anything**. (you probably don’t want this)
  - If you want the counter to count 100 steps per cycle (0 – 99), you write 99 to the ARR.
  - Get in the habit of using an expression when you write the value. e.g.:

```
LDR R0, =TIM6
LDR R1, =48-1

STR R1,[R0,#PSC]
```

Use TIM6 for periodic interrupt

- TIM6 is a subsystem, so you must (first) tell the RCC to enable its clock.
- Set its PSC and ARR values.
- Configure DIER to generate an interrupt on update.
- Enable the counter with the TIM6 control register.
- Unmask the interrupt.
- Once done, the ISR should be invoked 48M / (PSC+1) / (ARR +1) times per second.

Enable the RCC clock for TIM6

- Finding out how to enable the clock to a subsystem is an art.
  - Check Chapter 7 of the FRM.
  - Look at table 7.4.15 on page 142. (Register Map)
  - Search for TIM6EN in one of AHBENR, APB1ENR, and APB2ENR.
- TIM6EN is bit 4 of APB1ENR.

```
ldr  r0,=RCC
ldr  r1,[R0,#RCC_APB1ENR]
ldr  r2,=TIM6EN
orrs r1,r2
str  r1,[r0,#RCC_APB1ENR]
```

Set the PSC and ARR values

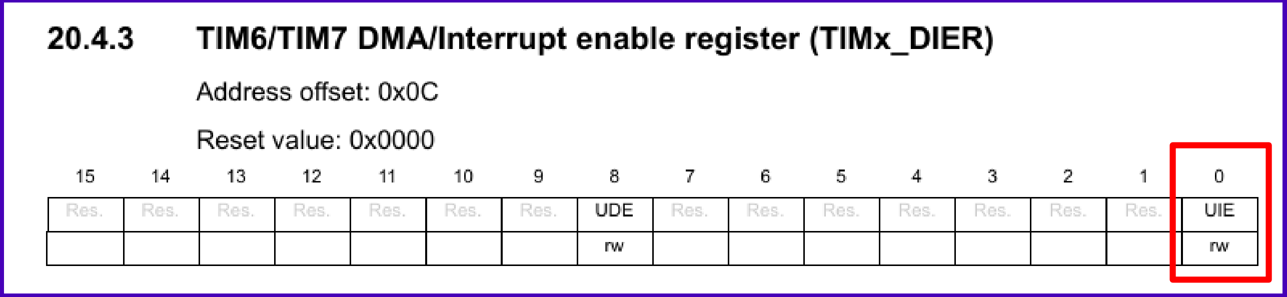
- Set PSC and ARR to one lower than you want.
- Example: If you want an update event to occur once every 10 seconds, you could say:
  - TIM6_PSC = 48000 – 1
  - TIM6_ARR = 10000 – 1
- OR…
  - TIM6_PSC = 24000 – 1
  - TIM6_ARR = 20000 – 1
- For reasons of power savings, it is better to set a larger prescaler value, so that other counters in the system run at a lower rate.

```
ldr   r0,=TIM6
ldr   r1,=48000-1
str   r1,[r0,#TIM_PSC]

ldr   r1,=1000-1
str   r1,[r0,#TIM_ARR]
```

Set DIER to enable intr on update

- The update event can be used to generate an interrupt.
- Set the UIE (Update Interrupt Enable) bit in the DIER (DMA/Interrupt Enable Register)



20.4.3 **TIM6/TIM7 DMA/Interrupt enable register (TIMx_DIER)**

Address offset: 0x0C

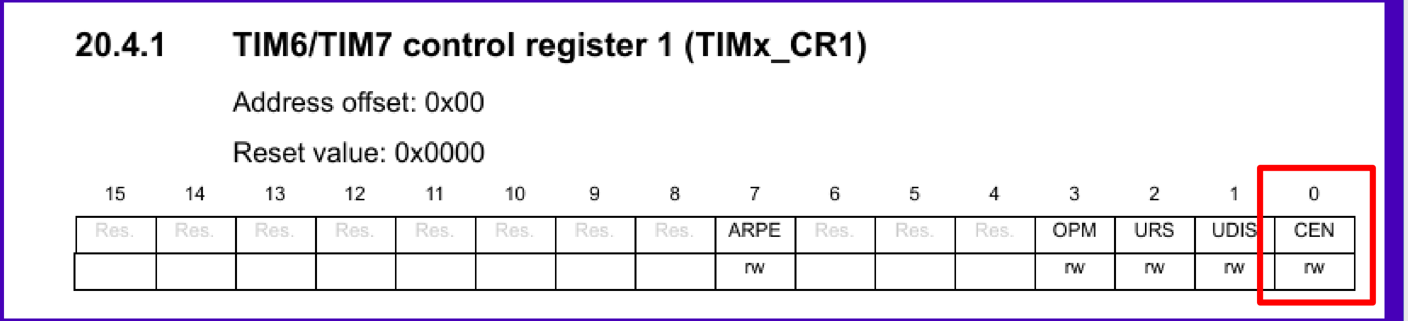
Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	UDE	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UIE
							rw								rw

```
ldr  r0,=TIM6
ldr  r1,[r0,#TIM_DIER]
ldr  r2,=TIM_DIER_UIE
orrs r1,r2
str  r1,[r0,#TIM_DIER]
```

Enable the Counter

- The control register has a counter enable bit:



20.4.1 TIM6/TIM7 control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ARPE	Res.	Res.	Res.	OPM	URS	UDIS	CEN
								rw				rw	rw	rw	rw

```
ldr  r0,=TIM6
ldr  r1,[r0,#TIM_CR1]
ldr  r2,=TIM_CR1_CEN
orrs r1,r2
str  r1,[r0,#TIM_CR1]
```

- The CEN bit enables the counter.
- Make sure you do this after setting PSC and ARR.

And unmask the interrupt

- Look up the symbolic name or interrupt number.
  - Write a bit to the NVIC_ISER to unmask the interrupt.
  - See Table 37, page 217 of FRM. TIM6_DAC is bit 17.

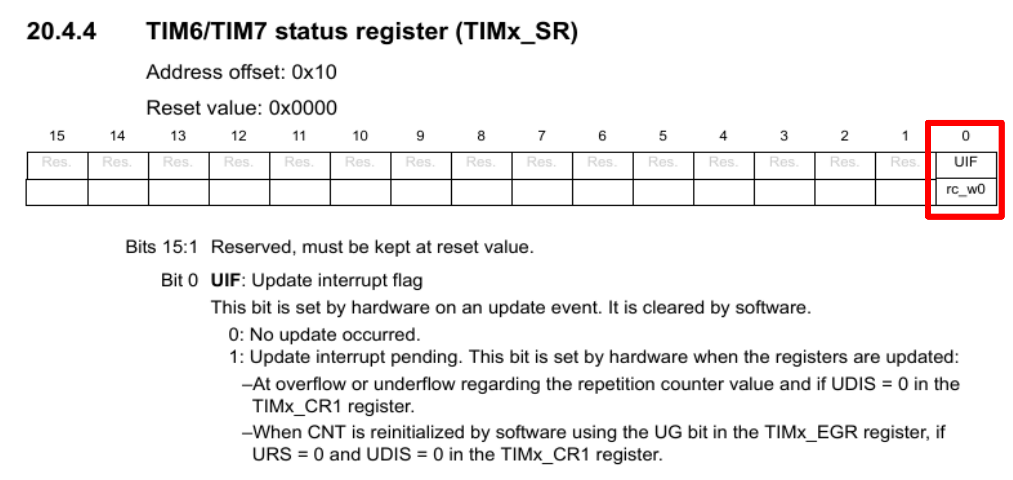
```
ldr  r0,=NVIC
ldr  r1,=NVIC_ISER
ldr  r2,=(1<<TIM6_DAC_IRQn)
str  r2,[r0,r1]
```

Write the Interrupt Service Routine

- Look up the exact name of the ISR.
  - It’s good to copy/paste it from startup_stm32.s
  - The name of the ISR is TIM6_DAC_IRQHandler
    - because it also does things with the Digital-to-Analog Converter subsystem. (We’ll learn about that soon.)

Acknowledge the Interrupt

- Every interrupt except SysTick must be acknowledged so that the ISR is not immediately re-invoked upon return.
- To acknowledge the interrupt for the TIM6 update interrupt, clear the UIF bit in the status register.



20.4.4 TIM6/TIM7 status register (TIMx_SR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	UIF
															rc_w0

Bits 15:1 Reserved, must be kept at reset value.

Bit 0 **UIF**: Update interrupt flag

This bit is set by hardware on an update event. It is cleared by software.

0: No update occurred.

1: Update interrupt pending. This bit is set by hardware when the registers are updated:

–At overflow or underflow regarding the repetition counter value and if UDIS = 0 in the TIMx_CR1 register.

–When CNT is reinitialized by software using the UG bit in the TIMx_EGR register, if URS = 0 and UDIS = 0 in the TIMx_CR1 register.

```
.type TIM6_DAC_IRQHandler, %function
.global TIM6_DAC_IRQHandler
TIM6_DAC_IRQHandler:
ldr   r0,=TIM6
ldr   r1,[r0,#TIM_SR1]   // read status reg
ldr   r2,=TIM_SR_UIF
bics r1,r2               // turn off UIF
str   r1,[r0,#TIM_SR]    // write it
```

TIM6 advantages

- TIM6 is much like SysTick.
  - Certainly more complicated
  - Much more flexible
    - Prescaler and counter allow a periodic interrupt rate between 48 MHz and 48,000,000 / 2^{32} (Once every 89.5 sec.)
  - Every one of the 9 timers in the STM32F091 can be used as a general purpose timer.
    - If you look up how to enable the other timers’ RCC clocks, and the base addresses of their control registers, you can use all the others in exactly the same way.
    - We’ll look at more advanced features of other timers later.

Event-Driven Programs

- Now that we can use timers and interrupts, it is possible to write programs that are entirely ***event-driven***. Every subroutine is invoked by an interrupt.
- We no longer need the `main()` program to do anything other than set up the timers.
  - One way to do nothing is have an endless loop.
  - A better way is to use the WFI instruction in a loop.

WFI: Wait For Interrupt

- The WFI instruction stops CPU execution and also puts the system into ***sleep mode***. The result is reduced power consumption.
- WFI finishes any time an interrupt service routine is invoked. (The ISR will return to the instruction following the WFI.)
- Use an endless loop of WFI:

```
endless:
    wfi
    b endless
```