Exam Micro controlleur

GPIOx -> 3min30

```
RCC->APB2ENR |= RCC_APB2ENR_IOPAEN; // activer la clock GPIOA
GPIOA->CRL &= ~(0b1111<<24);        // reset la config
GPIOA->CRL |= 0b1<<24;              // on passe en output push pull
```

TIMERx -> 15min

toutes les secondes de 0 à 40000 :

```
RCC->APB1ENR |= RCC_APB1ENR_TIM2EN; //activer la clock TIM1
TIM2->ARR = 40000;                  // ARR il compte de 0 à 40000
TIM2->PSC = 1799;                   // PSC : se calcule cf formule doc
TIM2->CR1 |= TIM_CR1_CEN;           // démarrer le timer
```

$periode_{timer} = periode_{horloge} \times (PSC + 1) \times (ARR + 1)$
$\frac{1}{1000} = \frac{1}{72Mhz} * (PSC + 1) * (ARR + 1)$ -> on fixe l'ARR d'abord
on prend ARR à 500 donc $PSC = \frac{72Mhz}{40001*1000} - 1 = 1799$

Interruption Timer -> 6min

> **NE PAS UTILISER TIMER1 C'EST DE LA MERDE**

```
void TIM2_IRQHandler() {
        TIM2->SR &= ~TIM_SR_UIF;    // remettre le flag d'interruption à 0
}
...
TIM2->DIER |= TIM_DIER_UIE;     // Activer l'interruption sur le TIM2
NVIC_EnableIRQ(TIM2_IRQn);      // Activer l'interruption via le NVIC
NVIC_SetPriority(TIM2_IRQn, 3); // Set la priorité de l'interruption (ex:3)
```

Interruption GPIO -> ∞ (toujours incompréhensible)

```
void EXTI15_10_IRQHandler() { /* code du handler */ }
...
RCC->APB2ENR |= RCC_APB2ENR_AFIOEN;           // activer l'AFIO
AFIO->EXTICR[3] |= AFIO_EXTICR4_EXTI13_PC;
```

```
EXTI->IMR |= EXTI_IMR_MR13;
EXTI->RTSR |= EXTI_RTSR_TR13;
```

PWM -> 15min

> **NE PAS UTILISER TIMER1 C'EST DE LA MERDE**
> -> bien mettre en alternate output push pull

```
// Mettre une PWN : channel 1 du TIM2 -> A0
GPIOA->CRL &= ~(0b1111);        // PA0 : reset mode
GPIOA->CRL |= 0b1001;           // PA0 : on passe en alternate output pp
TIM2->CCER |= TIM_CCER_CC1E;    // Activer le channel 1 de TIM2
TIM2->CCMR1 |= TIM_CCMR1_OC1M;  // Output compare 1 mode ????
TIM2->CCR1 = 10000;             // la moitié de l'ARR si on veut 50
```

ADC -> 30min

```
RCC->APB2ENR |= RCC_APB2ENR_ADC1EN;
RCC->CFGR |= RCC_CFGR_ADCPRE_DIV6;
ADC1->CR2 |= ADC_CR2_ADON;
ADC1->CR2 |= ADC_CR2_EXTTRIG;
ADC1->SQR3 |= 0xA;
ADC1->CR2 |= ADC_CR2_EXTSEL_0 | ADC_CR2_EXTSEL_1 | ADC_CR2_EXTSEL_2 ;
```

UART : reception -> 5min

$USARTDIV = \frac{f_{horloge}}{baudrate*16} = 234,375$ avec $f_{horloge}$ la fréquence de PCLK1
0,375 en hexa ça fait 6/16 donc 0x6 donc USARTDIV = OxEA6

```
void USART2_IRQHandler() {
        b = USART2->DR & 0xFF;              // lire la reception
        /* code du handler */
        USART2->SR &= ~(USART_SR_RXNE);     // remettre le flag à 0
}
...
RCC->APB1ENR |= RCC_APB1ENR_USART2EN;  // activer la clock UART
USART2->BRR |= 0xEA6;                  // see "USARTDIV" dans la doc
USART2->CR1 |= USART_CR1_UE;           // on active l'UART
USART2->CR1 |= USART_CR1_RE;           // on active la reception
USART2->CR1 |= USART_CR1_RXNEIE;       // interruption sur reception
```

```
NVIC_EnableIRQ(USART2_IRQn);           // configuration interruption
NVIC_SetPriority(USART2_IRQn,3);
```

UART : emission -> 5min

```
RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
USART2->BRR |= 0xEA6;
USART2->CR1 |= USART_CR1_TE;              // activer la transmission
GPIOA->CRL &= ~(0b1111<<8);
GPIOA->CRL |= (0b1001<<8);                // alternate output pp sur le tx
...
while ((USART2->SR&USART_SR_TXE) == 0) {} // attendre le droit de write
USART->DR = 'a';                          // ecrire a dans
```