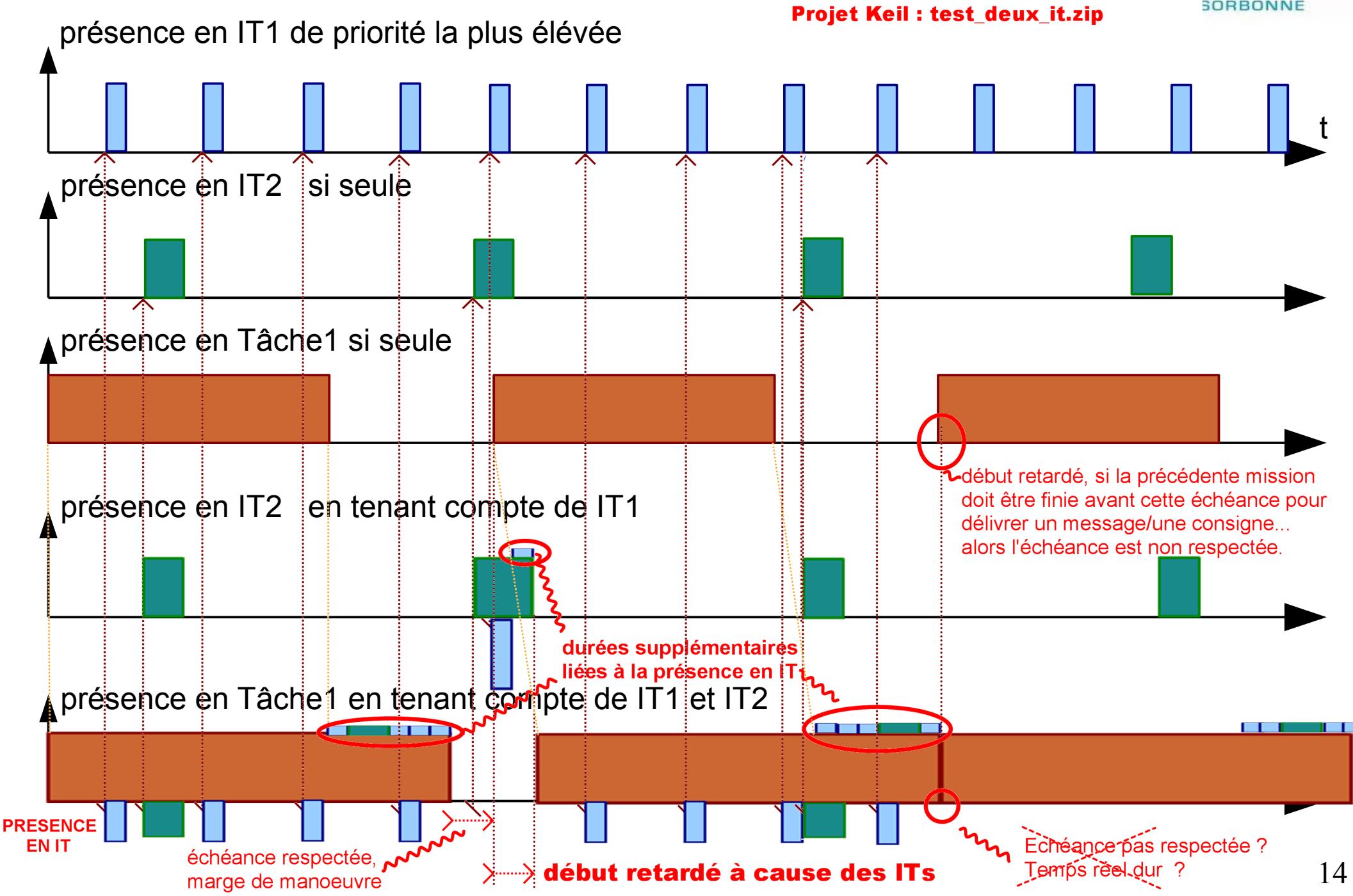
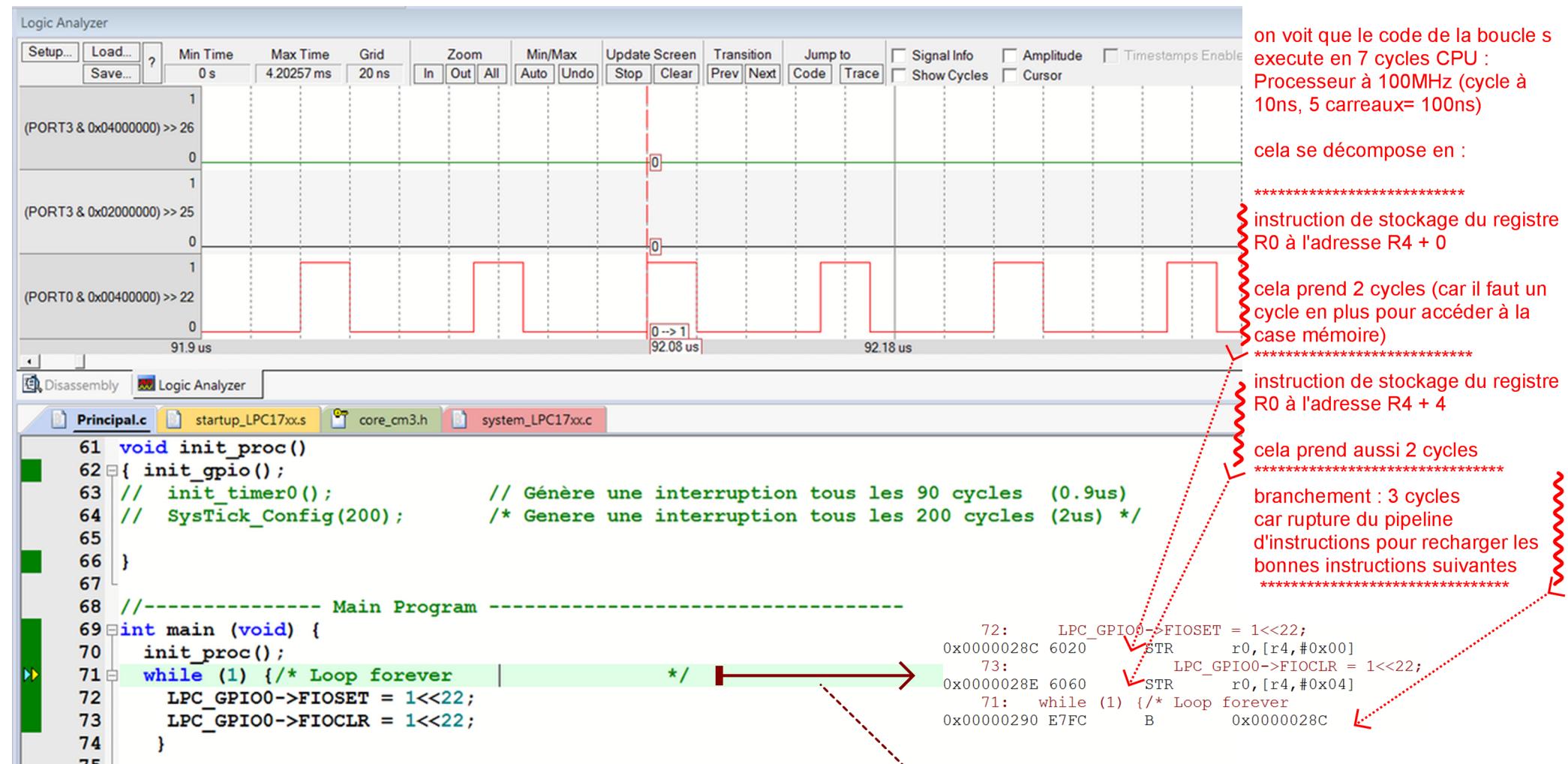


exemple de charge CPU



Etude sur keil : visualisation de chronogrammes à l'aide du simulateur.

programme simple sans IT :



les lignes init_timer0() et SysTick_Config(200) sont commentées, il n'y aucune interruption activée

La boucle While(1) du main a été optimisée par le compilateur
il a préparé deux registres avant de rentrer dans la boucle :
R4 contient l'adresse de LPC_GPIO0 -> FIOSET (0x2009C018)
R0 contient la valeur 1<<22

```

0x0000027C F24C0418 MOVW      r4,#0xC018
0x00000280 F2C20409 MOVT      r4,#0x2009
0x00000288 F44F0080 MOV       r0,#0x400000
    
```

Remarque : FIOCLR est décalé de 4 par rapport à FIOSET (c'est le 32 bits suivant)

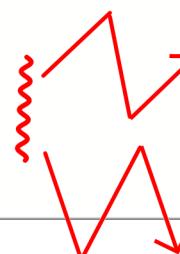
Complément de cours :

Regarder comment le compilateur passe du code C au code assembleur en lisant le document sur la création d'un nouveau projet en KEIL 5 :

- rôle et contenu du fichier LPC17xx.h
- étape de précompilation : générer un fichier .i
- niveaux de compilation
- fichier .map : le comprendre et l'exploiter

Programme avec deux interruptions activées : priorité 31 pour SysTick, priorité 31 pour Timer0 (PRIORITES IDENTIQUES)

```
void init_gpio(void)
{ LPC_SC->PCONP |= (1 << 15); /* power ON GPIO & IOCON (déjà par défaut) */
  LPC_GPIO3->FIODIR |= 0x03<<25; //P3.25 et P3.26 en sortie
  LPC_GPIO0->FIODIR |= 0x01<<22; //P0.22 en sortie
}
//-----
void init_timer0()
{ LPC_SC->PCONP |= (1 << 1); /*power on sur timer0 , déjà sous tension par défaut */
  // vérifier dans system_LPC17xx.C (configuration Wizard) que l'horloge périphérique
  // est bien sélectionnée sur CCLK/1 (menu clock configuration, registre PCLKSEL0)
  LPC_TIM0->TCR =0x03;
  LPC_TIM0->CTCR =0x00;
  LPC_TIM0->MR0 =120; //120 cycles avant IT;
  LPC_TIM0->MCR = 0x03;
  NVIC_EnableIRQ(TIMER0_IRQn) ;
  NVIC_SetPriority(TIMER0_IRQn, 31); // 31 priorité la plus faible , 0 la plus élevée
  LPC_TIM0->TCR =0x01;
}
//-----
void init_proc()
{ init_gpio();
  init_timer0(); // Génère une interruption tous les 90 cycles (0.9us)
  SysTick_Config(200); /* Genere une interruption tous les 200 cycles (2us) */
}
//----- Main Program -----
int main (void) {
  init_proc();
  while (1) /* Loop forever
    LPC_GPIO0->FIOSET = 1<<22;
    LPC_GPIO0->FIOCLR = 1<<22;
  }
}
```



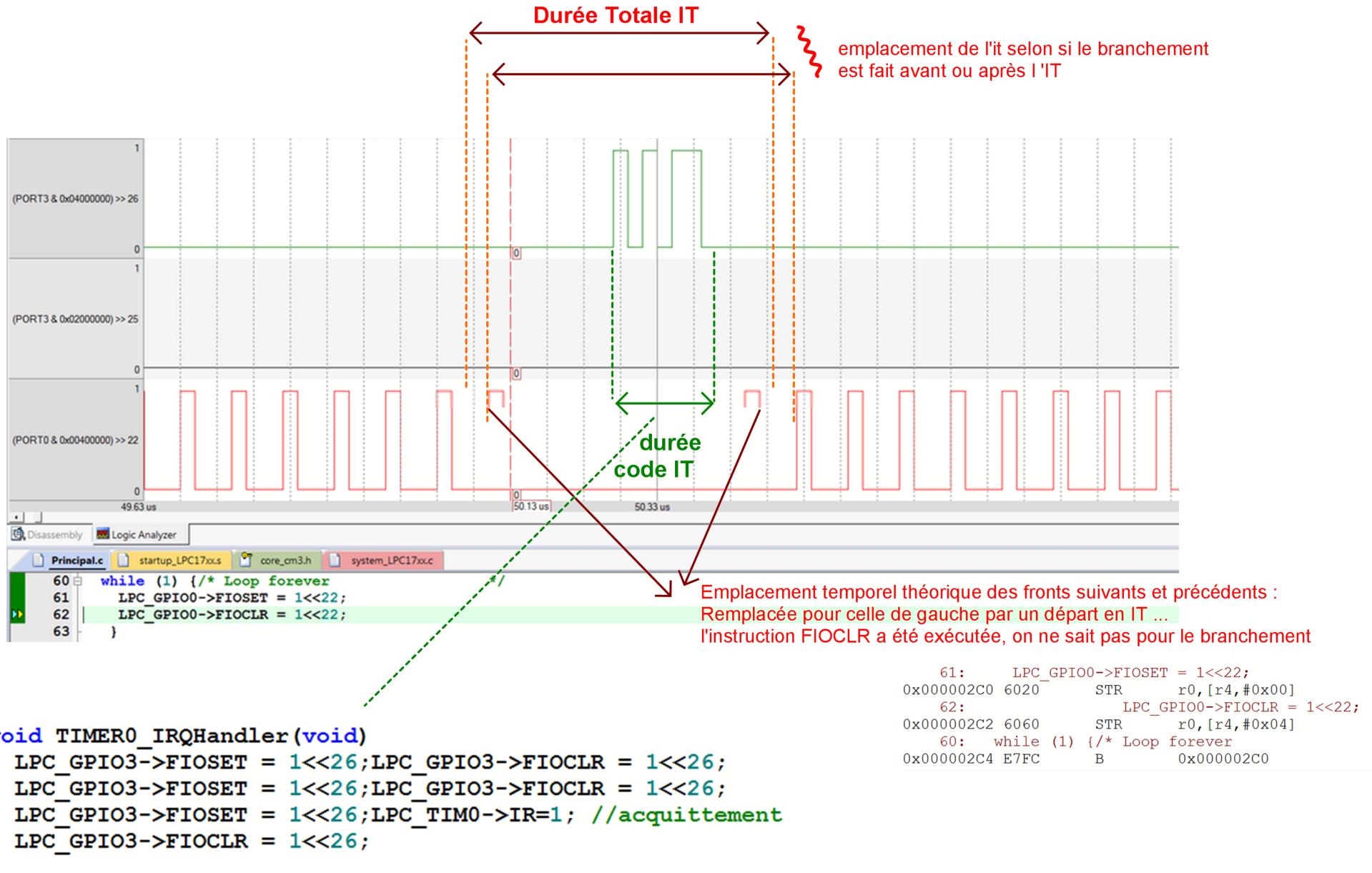
```
void TIMER0_IRQHandler(void)
{ LPC_GPIO3->FIOSET = 1<<26;LPC_GPIO3->FIOCLR = 1<<26;
  LPC_GPIO3->FIOSET = 1<<26;LPC_GPIO3->FIOCLR = 1<<26;
  LPC_GPIO3->FIOSET = 1<<26;LPC_TIM0->IR=1; //acquittement
  LPC_GPIO3->FIOCLR = 1<<26;
}

void SysTick_Handler (void) {
  LPC_GPIO3->FIOSET = 1<<25; LPC_GPIO3->FIOCLR = 1<<25;
  LPC_GPIO3->FIOSET = 1<<25; LPC_GPIO3->FIOCLR = 1<<25;
  LPC_GPIO3->FIOSET = 1<<25; LPC_GPIO3->FIOCLR = 1<<25;
  LPC_GPIO3->FIOSET = 1<<25; LPC_GPIO3->FIOCLR = 1<<25;
}
```

Quand l'évènement déclenchant l'IT survient :

- le programme principal est interrompu
- le code de l'interruption est exécuté
- Ensuite le programme principal est repris

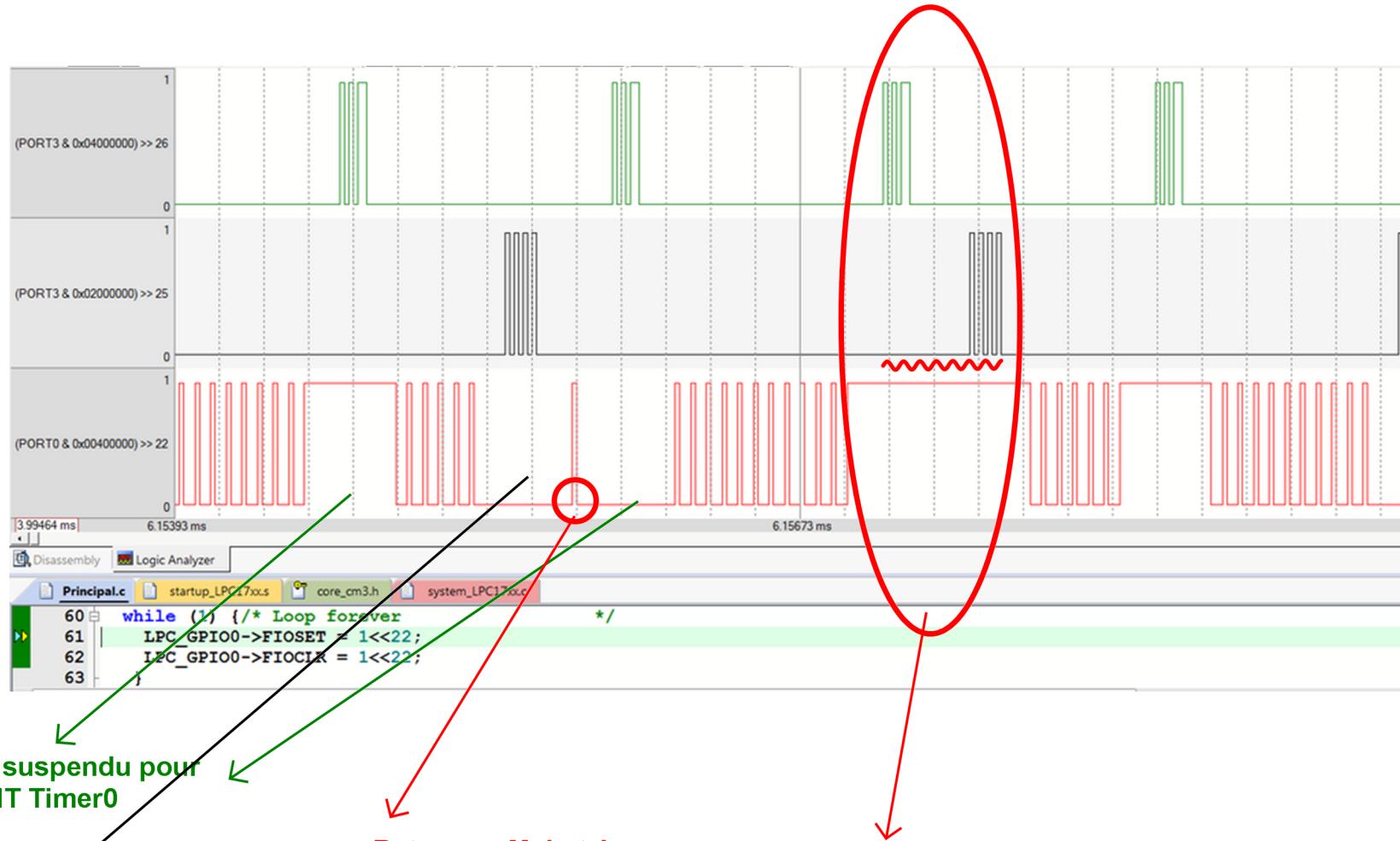
Exemple d'interruption : main interrompu pour servir l'IT Timer0



ON CONSTATE UN DELAI AVANT D'EXECUTER LE CODE DE L'IT, ET AVANT DE RENDRE LA MAIN AU MAIN()

CODE DU MAIN() ... SAUVEGARDE DE CONTEXTE ... CODE DE L'IT ... RESTITUTION DE CONTEXTE RETOUR AU MAIN()

priorité 31 pour SysTick, priorité 31 pour Timer0 (PRIORITES IDENTIQUES)



Main() suspendu pour servir IT Timer0

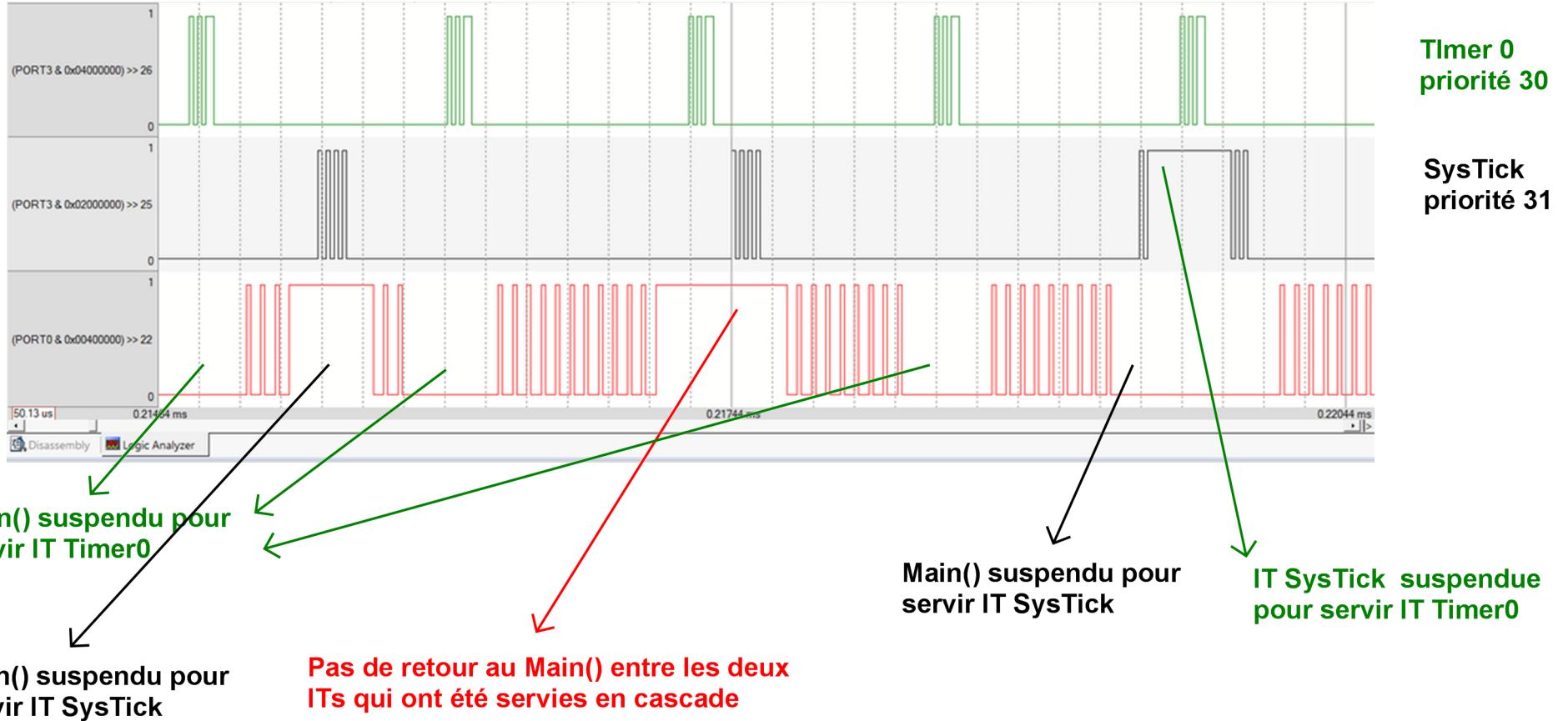
Retour au Main très très court

Pas de retour au Main() entre les deux ITs qui ont été servies en cascade

Main() suspendu pour servir IT SysTick

Les deux interruptions sont toutes deux de même priorité basse (priorité 31 la plus basse, 0 la plus haute), elles ne s'interrompront jamais l'une l'autre, celle qui arrive en second est différée jusqu'à la fin de l'autre. Si la fréquence des ITs était encore plus élevée, on n'aurait jamais le temps de revenir au main()

priorité 31 (la plus basse) pour SysTick,
 priorité 30 (la plus haute) pour Timer0
 (PRIORITES DIFFERENTES)



Quand deux interruptions de priorité différentes sont présentes dans un code :

- Les deux sont en capacité de suspendre l'exécution du Main() et lui rendre la main ensuite
- Celle de plus basse priorité sera différée si elle arrive pendant l'exécution d'une interruption de priorité haute, elles sont exécutées en cascade sans rendre la main au Main()
- Celle de plus basse priorité peut être interrompu par une interruption de plus haute priorité