Allocation d'interruption

Traduction Google ...

Aperçu

L'ESP32 a deux cœurs, avec 32 interruptions chacun. Chaque interruption a un certain niveau de priorité, la plupart (mais pas toutes) les interruptions sont connectées au mux d'interruption.

Parce qu'il y a plus de sources d'interruption que d'interruptions, il est parfois logique de partager une interruption dans plusieurs pilotes. L' esp_intr_alloc()abstraction existe pour masquer tous ces détails d'implémentation.

Un pilote peut allouer une interruption pour un certain périphérique en appelant esp_intr_alloc()(ou esp_intr_alloc_intrstatus()). Il peut utiliser les drapeaux passés à cette fonction pour définir le type d'interruption alloué, en spécifiant un niveau particulier ou une méthode de déclenchement. Le code d'allocation d'interruption trouvera alors une interruption applicable, utilisera le mux d'interruption pour le connecter au périphérique et y installera le gestionnaire d'interruption et l'ISR donnés.

Ce code présente deux types d'interruptions différents, gérés différemment : les interruptions partagées et les interruptions non partagées. Les plus simples sont les interruptions non partagées : une interruption distincte est allouée par esp_intr_alloc()appel et cette interruption est uniquement utilisée pour le périphérique qui lui est attaché, avec un seul ISR qui sera appelé. D'autre part, les interruptions partagées peuvent être déclenchées par plusieurs périphériques, plusieurs ISR étant appelés lorsque l'un des périphériques connectés signale une interruption. Ainsi, les ISR destinés aux interruptions partagées doivent vérifier l'état d'interruption du périphérique qu'ils desservent afin de vérifier si une action est requise.

Les interruptions non partagées peuvent être déclenchées par niveau ou front. Les interruptions partagées ne peuvent être que des interruptions de niveau en raison du risque d'interruptions manquées lorsque des interruptions de front sont utilisées.

Par exemple, disons que DevA et DevB partagent une interruption. DevB signale une interruption, donc la ligne INT passe au niveau haut. Le gestionnaire ISR appelle le code pour DevA mais ne fait rien. Ensuite, le gestionnaire ISR appelle le code pour DevB, mais ce faisant, DevA signale une interruption. L'ISR de DevB est terminé, il efface l'état d'interruption pour DevB et quitte le code d'interruption. Maintenant, une interruption pour DevA est toujours en attente, mais parce que la ligne INT n'est jamais descendue, comme DevA l'a maintenue haute même lorsque l'interruption pour DevB a été effacée, l'interruption n'est jamais desservie. Problèmes multicœurs

Les périphériques pouvant générer des interruptions peuvent être divisés en deux types :

- Périphériques externes, dans l'ESP32 mais en dehors des cœurs Xtensa eux-mêmes. La plupart des périphériques ESP32 sont de ce type.
- Périphériques internes, faisant partie des cœurs du processeur Xtensa eux-mêmes.

La gestion des interruptions diffère légèrement entre ces deux types de périphériques. Interruptions périphériques internes

Chaque cœur de processeur Xtensa possède son propre ensemble de six périphériques internes :

- Trois comparateurs de minuterie
- Un moniteur de performances
- Deux interruptions logicielles.

Les sources d'interruption internes sont définies dans esp_intr_alloc.h en tant que ETS_INTERNAL_*_INTR_SOURCE.

Ces périphériques ne peuvent être configurés qu'à partir du cœur auquel ils sont associés. Lors de la génération d'une interruption, l'interruption qu'ils génèrent est câblée à leur cœur associé ; il n'est pas possible, par exemple, qu'un comparateur de temporisation interne d'un cœur génère une interruption sur un autre cœur. C'est pourquoi ces sources ne peuvent être gérées qu'à l'aide d'une tâche exécutée sur ce noyau spécifique. Les sources d'interruption internes sont toujours attribuables en utilisant esp_intr_alloc()normalement, mais elles ne peuvent pas être partagées et auront toujours un niveau d'interruption fixe (à savoir, celui associé au matériel avec le périphérique). Interruptions périphériques externes

Les sources d'interruption restantes proviennent de périphériques externes. Ceux-ci sont définis dans soc/soc.h comme ETS_*_INTR_SOURCE.

Les emplacements d'interruption non internes dans les deux cœurs de processeur sont câblés à un multiplexeur d'interruption, qui peut être utilisé pour acheminer n'importe quelle source d'interruption externe vers l'un de ces emplacements d'interruption.

- L'allocation d'une interruption externe l'allouera toujours sur le cœur qui effectue l'allocation.
- La libération d'une interruption externe doit toujours se produire sur le même cœur sur lequel elle a été allouée.
- La désactivation et l'activation des interruptions externes à partir d'un autre cœur sont autorisées.
- Plusieurs sources d'interruption externes peuvent partager un slot d'interruption en le passant ESP_INTR_FLAG_SHAREDcomme indicateur à esp_intr_alloc().

Des précautions doivent être prises lors de l'appel esp_intr_alloc()à partir d'une tâche qui n'est pas épinglée à un noyau. Lors du changement de tâche, ces tâches peuvent migrer entre les cœurs. Par conséquent, il est impossible de dire sur quel processeur l'interruption est allouée, ce qui rend difficile la libération du handle d'interruption et peut également entraîner des difficultés de débogage. Il est conseillé de l'utiliser xTaskCreatePinnedToCore()avec un argument CoreID spécifique pour créer des tâches qui alloueront des interruptions. Dans le cas de sources d'interruption internes, cela est nécessaire. Gestionnaires d'interruptions IRAM-Safe

L' ESP_INTR_FLAG_IRAMindicateur enregistre un gestionnaire d'interruption qui s'exécute toujours à partir de l'IRAM (et lit toutes ses données à partir de la DRAM), et n'a donc pas besoin d'être désactivé pendant les opérations d'effacement et d'écriture flash.

Ceci est utile pour les interruptions qui nécessitent une latence d'exécution minimale garantie, car les opérations d'écriture et d'effacement flash peuvent être lentes (les effacements peuvent prendre des dizaines ou des centaines de millisecondes).

Il peut également être utile de conserver un gestionnaire d'interruptions dans IRAM s'il est appelé très fréquemment, pour éviter les ratés du cache flash.

Reportez-vous à la documentation de l'API flash SPI pour plus de détails. Plusieurs gestionnaires partageant une source

Plusieurs gestionnaires peuvent être affectés à une même source, étant donné que tous les gestionnaires sont alloués à l'aide du ESP_INTR_FLAG_SHAREDdrapeau. Ils seront tous affectés à l'interruption, à laquelle la source est attachée, et appelés séquentiellement lorsque la source est active. Les gestionnaires peuvent être désactivés et libérés individuellement. La source est attachée à l'interruption (activée), si un ou plusieurs gestionnaires sont activés, sinon détachée. Un gestionnaire ne sera jamais appelé lorsqu'il est désactivé, tandis que sa source peut toujours être déclenchée si l'un de ses gestionnaires est activé.

Les sources attachées à une interruption non partagée ne prennent pas en charge cette fonctionnalité.

Bien que le framework prenne en charge cette fonctionnalité, vous devez l'utiliser avec beaucoup de prudence . Il existe généralement deux manières d'empêcher le déclenchement d'une interruption : désactiver la source ou masquer l'état de l'interruption périphérique . IDF ne gère que l'activation et la désactivation de la source elle-même, laissant les bits d'état et de masque à la charge des utilisateurs. Les bits d'état doivent soit être masqués avant que le gestionnaire qui en est responsable ne soit désactivé, soit être masqués puis correctement traités dans une autre interruption activée . Veuillez noter que le fait de laisser certains bits d'état non gérés sans les masquer, tout en désactivant les gestionnaires correspondants, entraînera le déclenchement indéfini de la ou des interruptions, entraînant ainsi un plantage du système.

Référence API

En tête de fichier

- composants/esp_hw_support/include/esp_intr_alloc.h

Les fonctions

esp_err_t esp_intr_mark_shared (int intno , int cpu , bool is_in_iram)

- Marquez une interruption comme une interruption partagée.
- Cela marquera une certaine interruption sur le processeur spécifié comme une interruption pouvant être utilisée pour connecter des gestionnaires d'interruptions partagés.
- Paramètres

1. -intno - Le numéro de l'interruption (0-31)
2. -cpu - CPU sur lequel l'interruption doit être marquée comme partagée (0 ou 1)
3. -is_in_iram - L'interruption partagée est destinée aux gestionnaires qui résident dans l'IRAM et l'int peut être laissé activé pendant que le cache flash est désactivé.

ESP_ERR_INVALID_ARG si cpu ou intno est invalide ESP_OK sinon

esp_err_t esp_intr_reserve (int intno , int cpu)

- Réservez une interruption à utiliser en dehors de ce cadre.
- Cela marquera une certaine interruption sur le processeur spécifié comme réservée, à ne pas allouer pour quelque raison que ce soit.

Paramètres

- intno - Le numéro de l'interruption (0-31)
- cpu - CPU sur lequel l'interruption doit être marquée comme partagée (0 ou 1)
- ESP_ERR_INVALID_ARG si cpu ou intno est invalide ESP_OK sinon

esp_err_t esp_intr_alloc (int source , int flags , intr_handler_t handler , void * arg , intr_handle_t * ret_handle)

- Allouer une interruption avec les paramètres donnés.
- Cela trouve une interruption qui correspond aux restrictions indiquées dans le paramètre flags, y mappe la source d'interruption donnée et connecte également le gestionnaire d'interruption donné (avec un argument facultatif). Si nécessaire, il peut également renvoyer un handle pour l'interruption.
- L'interruption sera toujours allouée sur le noyau qui exécute cette fonction.
- Si l'indicateur ESP_INTR_FLAG_IRAM est utilisé et que l'adresse du gestionnaire n'est pas dans IRAM ou RTC_FAST_MEM, alors ESP_ERR_INVALID_ARG est renvoyé.

Paramètres

- source – La source d'interruption. L'une des sources de multiplexage d'interruption ETS_*_INTR_SOURCE, comme défini dans soc/soc.h, ou l'une des sources internes ETS_INTERNAL_*_INTR_SOURCE comme défini dans cet en-tête.
- flags – Un masque ORred de ESP_INTR_FLAG_* définit. Celles-ci limitent le choix des interruptions parmi lesquelles cette routine peut choisir. Si cette valeur est 0, il allouera par défaut une interruption non partagée de niveau 1, 2 ou 3. S'il s'agit de ESP_INTR_FLAG_SHARED, il allouera une interruption partagée de niveau 1. Le réglage ESP_INTR_FLAG_INTRDISABLED reviendra de cette fonction avec l'interruption désactivée .
- handler – Le gestionnaire d'interruptions. Doit être NULL lorsqu'une interruption de niveau >3 est demandée, car ces types d'interruptions ne sont pas appelables en C.
- arg - Argument facultatif pour passé au gestionnaire d'interruption
- ret_handle - Pointeur vers un intr_handle_t pour stocker un handle qui peut ensuite être utilisé pour demander des détails ou libérer l'interruption. Peut être NULL si aucun handle n'est requis.
- ESP_ERR_INVALID_ARG si la combinaison d'arguments est invalide. ESP_ERR_NOT_FOUND Aucune interruption libre trouvée avec les drapeaux spécifiés ESP_OK sinon

esp_err_t esp_intr_alloc_intrstatus (int source , int flags , uint32_t intrstatusreg , uint32_t intrstatusmask , intr_handler_t handler , void * arg , intr_handle_t * ret_handle)

Allouer une interruption avec les paramètres donnés.

- Cela fait essentiellement la même chose que esp_intr_alloc, mais permet de spécifier un combo registre et masque. Pour les interruptions partagées, le gestionnaire n'est appelé que si une lecture du registre spécifié, ANDed avec le masque, renvoie non nul. En transmettant une adresse de registre d'état d'interruption et un masque d'adaptation, cela peut être utilisé pour accélérer la gestion des interruptions dans le cas où une interruption partagée est déclenchée ; en vérifiant d'abord les statuts d'interruption, le code peut décider quels ISR peuvent être ignorés
- source – La source d'interruption. L'une des sources de multiplexage d'interruption ETS_*_INTR_SOURCE, comme défini dans soc/soc.h, ou l'une des sources internes ETS_INTERNAL_*_INTR_SOURCE comme défini dans cet en-tête.
- flags – Un masque ORred de ESP_INTR_FLAG_* définit. Celles-ci limitent le choix des interruptions parmi lesquelles cette routine peut choisir. Si cette valeur est 0, il allouera par défaut une interruption non partagée de niveau 1, 2 ou 3. S'il s'agit de ESP_INTR_FLAG_SHARED, il allouera une interruption partagée de niveau 1. Le réglage ESP_INTR_FLAG_INTRDISABLED reviendra de cette fonction avec l'interruption désactivée .
- intrstatusreg - L'adresse d'un registre d'état d'interruption
- intrstatusmask – Un masque. Si une lecture de l'adresse intrstatusreg a l'un des bits qui sont 1 dans le jeu de masques, l'ISR sera appelé. Sinon, il sera ignoré.
- handler – Le gestionnaire d'interruptions. Doit être NULL lorsqu'une interruption de niveau >3 est demandée, car ces types d'interruptions ne sont pas appelables en C.
- arg - Argument facultatif pour passé au gestionnaire d'interruption
- ret_handle - Pointeur vers un intr_handle_t pour stocker un handle qui peut ensuite être utilisé pour demander des détails ou libérer l'interruption. Peut être NULL si aucun handle n'est requis.
- ESP_ERR_INVALID_ARG si la combinaison d'arguments est invalide. ESP_ERR_NOT_FOUND Aucune interruption libre trouvée avec les drapeaux spécifiés ESP_OK sinon

esp_err_t esp_intr_free (intr_handle_t handle)

Désactiver et libérer une interruption.

- Utilisez un handle d'interruption pour désactiver l'interruption et libérer les ressources qui lui sont associées. Si le cœur actuel n'est pas le cœur qui a enregistré cette interruption, cette routine sera affectée au cœur qui a alloué cette interruption, bloquant et attendant jusqu'à ce que la ressource soit libérée avec succès.

```
  Lorsque le gestionnaire partage sa source avec d'autres
gestionnaires, les bits d'état d'interruption dont il est
responsable doivent être gérés correctement avant de le libérer.
voir esp_intr_disablepour plus de détails. Veuillez ne pas
appeler cette fonction dans esp_ipc_call_blocking.
```

Paramètres

- handle – Le handle, tel qu'obtenu par esp_intr_alloc ou esp_intr_alloc_intrstatus
- ESP_ERR_INVALID_ARG le handle est NULL ESP_FAIL n'a pas réussi à libérer ce handle ESP_OK sinon

int esp_intr_get_cpu (intr_handle_t handle)

Obtenir le numéro de CPU auquel une interruption est liée.

Paramètres

- handle – Le handle, tel qu'obtenu par esp_intr_alloc ou esp_intr_alloc_intrstatus

```
     Le numéro de cœur où l'interruption est allouée
```

int esp_intr_get_intno (intr_handle_t handle)

```
  Obtenir l'interruption allouée pour un certain handle.
```

```
  Paramètres
```

```
     handle – Le handle, tel qu'obtenu par esp_intr_alloc ou
esp_intr_alloc_intrstatus
  Retour
```

```
     Le numéro d'interruption
```

esp_err_t esp_intr_disable (intr_handle_t handle)

```
  Désactiver l'interruption associée au handle.
```

```
  Noter
```

```
     Pour les interruptions locales (sources ESP_INTERNAL_*), cette
fonction doit être appelée sur le CPU auquel l'interruption est allouée.
D'autres interruptions n'ont pas une telle restriction.
```

```
     Lorsque plusieurs gestionnaires partagent une même source
d'interruption, les bits d'état d'interruption, qui sont gérés dans le
gestionnaire à désactiver, doivent être masqués avant la désactivation ou
gérés correctement dans d'autres interruptions activées. L'absence de
gestion de l'état d'interruption entraînera des appels d'interruption
infinis et finalement un plantage du système.
```

```
  Paramètres
```

```
     handle – Le handle, tel qu'obtenu par esp_intr_alloc ou
esp_intr_alloc_intrstatus
```

```
  Retour

      ESP_ERR_INVALID_ARG si la combinaison d'arguments est invalide. ESP_OK
sinon
```

esp_err_t esp_intr_enable (intr_handle_t handle)

```
  Activez l'interruption associée au handle.

  Noter

  Pour les interruptions locales (sources ESP_INTERNAL_*), cette fonction
doit être appelée sur le CPU auquel l'interruption est allouée. D'autres
interruptions n'ont pas une telle restriction.

  Paramètres

      handle – Le handle, tel qu'obtenu par esp_intr_alloc ou
esp_intr_alloc_intrstatus
  Retour

      ESP_ERR_INVALID_ARG si la combinaison d'arguments est invalide. ESP_OK
sinon
```

esp_err_t esp_intr_set_in_iram (intr_handle_t handle , bool is_in_iram)

```
  Définissez le statut "in IRAM" du gestionnaire.

  Noter

  Ne fonctionne pas sur les interruptions partagées.

  Paramètres

          handle – Le handle, tel qu'obtenu par esp_intr_alloc ou
esp_intr_alloc_intrstatus

          is_in_iram – Indique si le gestionnaire associé à ce handle réside
dans IRAM. Les gestionnaires résidant dans IRAM peuvent être appelés lorsque
le cache est désactivé.

  Retour

      ESP_ERR_INVALID_ARG si la combinaison d'arguments est invalide. ESP_OK
sinon
```

void esp_intr_noniram_disable (void)

```
  Désactivez les interruptions qui ne sont pas spécifiquement marquées comme
```

```
s'exécutant à partir d'IRAM.
```

void esp_intr_noniram_enable (void)

```
  Réactivez les interruptions désactivées par esp_intr_noniram_disable.
```

void esp_intr_enable_source (int inum)

```
  activer la source d'interruption en fonction de son numéro

  Paramètres

      inum - numéro d'interruption de 0 à 31
```

void esp_intr_disable_source (int inum)

```
  désactiver la source d'interruption en fonction de son numéro

  Paramètres

      inum - numéro d'interruption de 0 à 31
```

statique en ligne int esp_intr_flags_to_level (drapeaux int)

```
  Obtenez le niveau d'interruption le plus bas à partir des drapeaux.

  Paramètres

      flags - Les mêmes drapeaux qui passent à l'
esp_intr_alloc_intrstatusAPI
```

Macros

ESP_INTR_FLAG_LEVEL1

```
  Indicateurs d'allocation d'interruption.

  Ces drapeaux peuvent être utilisés pour spécifier les qualités
d'interruption dont le code appelant esp_intr_alloc* a besoin. Accepter un
vecteur d'interruption de niveau 1 (priorité la plus basse)
```

ESP_INTR_FLAG_LEVEL2

```
  Acceptez un vecteur d'interruption de niveau 2.
```

ESP_INTR_FLAG_LEVEL3

```
  Acceptez un vecteur d'interruption de niveau 3.
```

ESP_INTR_FLAG_LEVEL4

```
  Acceptez un vecteur d'interruption de niveau 4.
```

ESP_INTR_FLAG_LEVEL5

```
  Acceptez un vecteur d'interruption de niveau 5.
```

ESP_INTR_FLAG_LEVEL6

```
  Acceptez un vecteur d'interruption de niveau 6.
```

ESP_INTR_FLAG_NMI

```
  Accepter un vecteur d'interruption de niveau 7 (priorité la plus élevée)
```

ESP_INTR_FLAG_SHARED

```
  L'interruption peut être partagée entre les ISR.
```

ESP_INTR_FLAG_EDGE

```
  Interruption déclenchée par le front.
```

ESP_INTR_FLAG_IRAM

```
  ISR peut être appelé si le cache est désactivé.
```

ESP_INTR_FLAG_INTRDISABLED

```
  Retour avec cette interruption désactivée.
```

ESP_INTR_FLAG_LOWMED

```
  Interruptions à priorité faible et moyenne. Ceux-ci peuvent être traités
en C.
```

ESP_INTR_FLAG_HIGH

```
  Interruptions de haut niveau. Doit être manipulé lors du montage.
```

ESP_INTR_FLAG_LEVELMASK

```
  Masque pour tous les drapeaux de niveau.
```

ETS_INTERNAL_TIMER0_INTR_SOURCE

```
  Source d'interruption du temporisateur de plate-forme 0.
```

```
  Les fonctions esp_intr_alloc* peuvent allouer un int pour toutes les
sources d'interruption ETS_*_INTR_SOURCE qui sont acheminées via le mux
```

```
d'interruption. Outre ces sources, chaque cœur possède également des sources
internes qui ne passent pas par le mux d'interruption. Pour allouer une
interruption à ces sources, passez ces pseudo-sources aux fonctions.
```

ETS_INTERNAL_TIMER1_INTR_SOURCE

```
  Source d'interruption de la minuterie 1 de la plate-forme.
```

ETS_INTERNAL_TIMER2_INTR_SOURCE

```
  Source d'interruption de la minuterie 2 de la plate-forme.
```

ETS_INTERNAL_SW0_INTR_SOURCE

```
  Logiciel int source 1.
```

ETS_INTERNAL_SW1_INTR_SOURCE

```
  Logiciel int source 2.
```

ETS_INTERNAL_PROFILING_INTR_SOURCE

```
  Int source pour le profilage.
```

ETS_INTERNAL_UNUSED_INTR_SOURCE

```
  L'interruption n'est affectée à aucune source.
```

ETS_INTERNAL_INTR_SOURCE_OFF

```
  Fournit à SystemView des ID IRQ positifs, sinon les événements du
planificateur ne s'affichent pas correctement
```

ESP_INTR_ENABLE (numéro)

```
  Activer l'interruption par numéro d'interruption
```

ESP_INTR_DISABLE (nombre)

```
  Désactiver l'interruption par numéro d'interruption
```

Définitions des types

typedef void (* intr_handler_t) (void * arg)

```
  Prototype de fonction pour la fonction de gestionnaire d'interruption
```

typedef struct intr_handle_data_t intr_handle_data_t

```
  Structure de données associée au gestionnaire d'interruptions
```

typedef intr_handle_data_t * intr_handle_t

```
  Handle vers un gestionnaire d'interruption
```