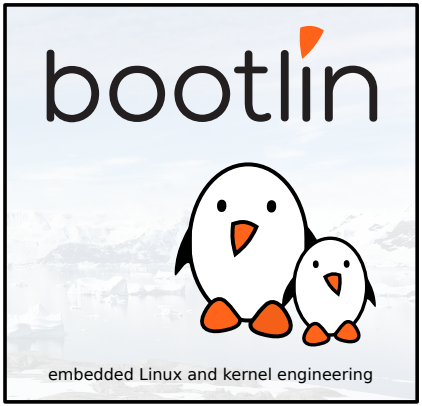
# V4L2 pour l'accélération matérielle de décodage vidéo

Paul Kocialkowski
*paul@bootlin.com*





# Paul Kocialkowski

- Ingénieur Linux Embarqué chez Bootlin
  - **Expertise** Linux Embarqué
  - **Développement**, consulting et formations
  - Fortement axé sur l'open-source
- Contributeur à l'Open-source
- Habitant à **Toulouse** en France

# Prise en charge du VPU Allwinner dans Linux *mainline*

- Objet du projet :
  - **Plateformes Allwinner** : SoCs ARMv7 et ARMv8
  - **VPU** : Accélérateur matériel de décodage vidéo
  - **Noyau Linux *mainline*** : intégration à l'arbre officiel
  - **Intégration** en espace utilisateur

# Prise en charge du VPU Allwinner dans Linux *mainline*

- Objet du projet :
  - **Plateformes Allwinner** : SoCs ARMv7 et ARMv8
  - **VPU** : Accélérateur matériel de décodage vidéo
  - **Noyau Linux *mainline*** : intégration à l'arbre officiel
  - **Intégration** en espace utilisateur
- Situation initiale :
  - Noyaux Allwinner (3.4/3.10) et blobs en espace utilisateur
  - Reverse Engineering par la communauté : **Cedrus**
  - Première implémentation : Stage chez Bootlin, été 2017

# Prise en charge du VPU Allwinner dans Linux *mainline*

- Objet du projet :
  - **Plateformes Allwinner** : SoCs ARMv7 et ARMv8
  - **VPU** : Accélérateur matériel de décodage vidéo
  - **Noyau Linux *mainline*** : intégration à l'arbre officiel
  - **Intégration** en espace utilisateur
- Situation initiale :
  - Noyaux Allwinner (3.4/3.10) et blobs en espace utilisateur
  - Reverse Engineering par la communauté : **Cedrus**
  - Première implémentation : Stage chez Bootlin, été 2017
- Déroulement :
  - Campagne de financement participatif
  - Stage de fin d'études
  - Période de 6 mois, Mars-Août 2018

# Compression vidéo

- Taille exorbitante des données brutes (1h30 en 1280x720, RGB32, 30 fps) :

$$t = 1280 \times 720 \times 3 \times 30 \times 90 \times 60 \simeq 417 \ Gio$$

# Compression vidéo

- Taille exorbitante des données brutes (1h30 en 1280x720, RGB32, 30 fps) :

$$t = 1280 \times 720 \times 3 \times 30 \times 90 \times 60 \simeq 417 \ Gio$$

- Méthodes de compression vidéo :
  - Compression visuelle : sous-échantillonnage YUV
  - Compression spatiale : DCT et filtrage
  - Compression temporelle : prédiction multi-directionnelle et interpolation
  - Compression entropique : Codage de Huffman, arithmétique (CABAC, CAVLC)

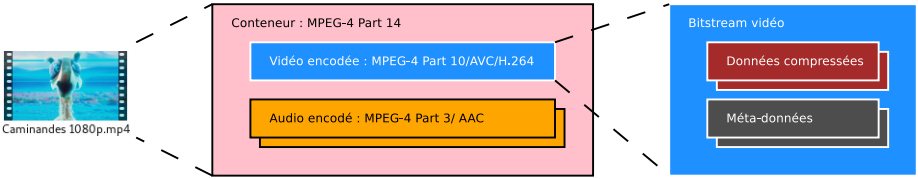
# Compression vidéo

- Taille exorbitante des données brutes (1h30 en 1280x720, RGB32, 30 fps) :

$$t = 1280 \times 720 \times 3 \times 30 \times 90 \times 60 \simeq 417 \ \textit{Gio}$$

- Méthodes de compression vidéo :
  - Compression visuelle : sous-échantillonnage YUV
  - Compression spatiale : DCT et filtrage
  - Compression temporelle : prédiction multi-directionnelle et interpolation
  - Compression entropique : Codage de Huffman, arithmétique (CABAC, CAVLC)
- Réduction drastique de la taille (facteur 610 pour 700 *Mio*)
- Opérations coûteuses à l'encodage et au décodage
- Nécessaire de décharger le CPU dans l'embarqué

# Structure et encapsulation d'une vidéo




- Données compressées :
  - *Macroblock ou Coding Tree Block : unité de base*
  - *Slice* : regroupement de MB/CTB
  - *Picture (frames/field)* : image reconstituée
- Méta-données :
  - Paramètres de la chaîne de décodage

**Bitstream : données compressées + méta-données (+ en-têtes)**

# Approches pour l'accélération du décodage et types de VPU

Chaîne logicielle : **stateful**

- Processeur auxiliaire (type DSP)
- Micrologiciel associé
- Passage du bitstream
- Commandes de décodage
- Gère le contexte de décodage

# Approches pour l'accélération du décodage et types de VPU

Chaîne logicielle : **stateful**

- Processeur auxiliaire (type DSP)
- Micrologiciel associé
- Passage du bitstream
- Commandes de décodage
- Gère le contexte de décodage

Chaîne matérielle : **stateless**

- Bloc matériel dédié
- Pas de micrologiciel
- Passage des slices
- Registres pour les méta-données
- Pas de contexte de décodage

Interface avec le CPU : MMIO

# API V4L2 : Généralités

Concepts principaux :

- **Queues** : `OUTPUT` (vers le matériel), `CAPTURE` (depuis le matériel)
- **Format** par queue : dimensions, *pixel format*
- **Buffers** tournants par queue : données
- **Contrôles** : méta-données de configuration

# API V4L2 : Généralités

Concepts principaux :

- **Queues** : `OUTPUT` (vers le matériel), `CAPTURE` (depuis le matériel)
- **Format** par queue : dimensions, *pixel format*
- **Buffers** tournants par queue : données
- **Contrôles** : méta-données de configuration

Opérations : `/dev/videoN`:

- Configuration : `VIDIOC_S_FMT`, `VIDIOC_S_CTRL`/`VIDIOC_S_EXT_CTRLS`
- Allocation/libération des buffers : `VIDIOC_REQBUF` ou `VIDIOC_QUERYBUF`
- Démarrage/fin du flux : `VIDIOC_STREAMON`/`VIDIOC_STREAMOFF`
- Ajout/retrait des buffers en queue : `VIDIOC_QBUF`/`VIDIOC_DQBUF`
- Attente des données : `poll`

# V4L2 M2M : VPU *stateful*

Application au décodage *stateful* :

- Utilise les deux queues simultanément : *Memory to Memory* (M2M)
- Queue `OUTPUT` : données du bitstream (e.g. `V4L2_PIX_FMT_H264`)
- Queue `CAPTURE` : images décodées (e.g. `V4L2_PIX_FMT_NV12`)

# V4L2 M2M : VPU *stateful*

Application au décodage *stateful* :

- Utilise les deux queues simultanément : *Memory to Memory* (M2M)
- Queue `OUTPUT` : données du bitstream (e.g. `V4L2_PIX_FMT_H264`)
- Queue `CAPTURE` : images décodées (e.g. `V4L2_PIX_FMT_NV12`)

Côté driver et framework V4L2 M2M :

- Contexte de décodage : associe les buffers
- Routine `device_run()` pour le décodage
- Complétion du décodage lié aux contexte (interruption)
- Planification du décodage dès les conditions réunies

# Media Request API : VPU *stateless*

Besoins spécifiques des VPU *stateless*:

- Passage des slices de **données compressées** : buffers
- Passage des **méta-données** : contrôles
- **Synchronisation** des buffers et des contrôles

**Problème : Pas de mécanisme de synchronisation explicite**

# Media Request API : VPU *stateless*

Besoins spécifiques des VPU *stateless*:

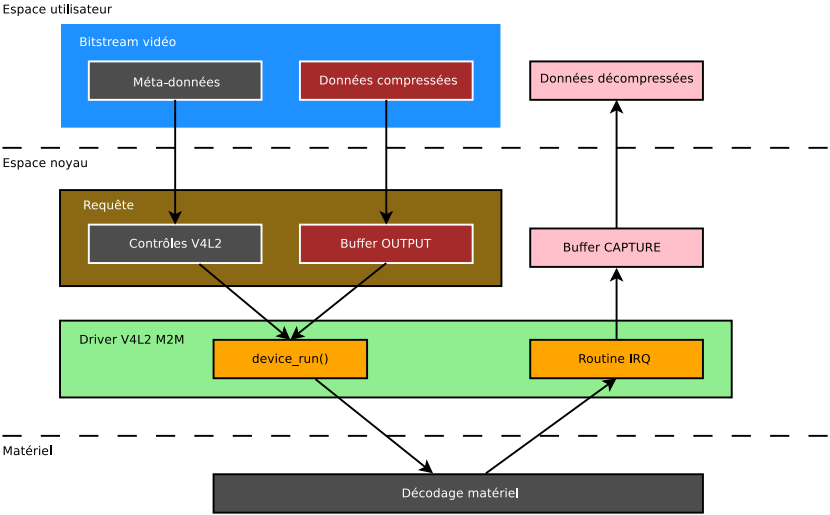
- Passage des slices de **données compressées** : buffers
- Passage des **méta-données** : contrôles
- **Synchronisation** des buffers et des contrôles

**Problème : Pas de mécanisme de synchronisation explicite**

Media Request API :

- Requêtes associées aux buffers et contrôles
- Ne prennent effet qu'au traitement de la requête
- Poursuite des opérations habituelles (M2M)

# Décodage avec V4L2 M2M et Request API en résumé



# Questions? Suggestions? Comments?

Paul Kocialkowski
*paul@bootlin.com*


https://bootlin.com/pub/conferences/