ARM-ADA 2.0

MANUEL TÉCHNIQUE

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE ET INFORMATIQUE

Juin 2020

Auteur : S. DI MERCURIO

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

TABLE DES MATIÈRES

INDEX DES FIGURES

INDEX DES TABLEAUX

1 - Présentation

Le projet ARM-ADA 2 est l'évolution du projet ARM-ADA initial. Il doit permettre l'enseignement de l'algorithmique et de la programmation, notamment en s'appuyant sur le langage ADA et un boîtier externe semblable à un console de jeu portative. L'étudiant développe son programme sur un PC, en utilisant des outils et bibliothèques logicielles soit existante (sur étagère) soit développés en interne, soit adapté en interne à partir d'un produit existant. L'étudiant exécute et dévermine ensuite son programme sur le boîtier externe, en suivant le plan pédagogique défini par l'enseignant.

1.1 - Rappels du projet initial

Le projet initial mettait en œuvre un boîtier architecturé autour d'un processeur STM32F303 auquel était ajouté un écran TFT 320x240 de 3 pouces, des capteurs IMU, des boutons et un amplificateur audio.

Le boîtier embarquait un logiciel résident ainsi qu'un bootloader pour permettre de reprogrammer le boîtier. Le programme de l'étudiant était téléchargé dans le boîtier à coté du logiciel résident et s'appuyait sur les routines fournies par ce résident pour accéder aux ressources du boîtier, créant un couplage fort entre l'application et le système résident.

De plus, le déverminage n'était pas possible, rendant la mise au point d'un programme plus compliquée.

Enfin, le choix du langage de programmation était imposé et uniquement ADA.

1.2 - Évolutions apportées par la nouvelle solution

Le nouveau projet s'appuie sur un boîtier architecturé autour d'une carte de développement 32F746-EVAL, basée sur un microcontrôleur STM32F746NG et embarquant :

- un écran TFT de 5 pouces (480x272), tactile (capacitif)
- 16 Mo de Flash externe (pour les ressources graphiques)
- 4 Mo de RAM externe
- une connecteur Ethernet 100Mbits
- un codec audio
- 2 microphones mems en façade

A cela sont rajoutés :

- un ensemble accéléromètre, gyroscope et magnétomètre 3D
- un capteur de pression absolue
- et un contrôleur de bouton

Côté logiciel et outils, l'évolution la plus marquante concerne le programme résident dans le boîtier et le couplage avec l'application développée par l'étudiant. En effet, le

programme résident ne sert plus que de lanceur d'application et ne partage plus aucune routine avec le programme développée en TP.

Le logiciel développée par l'étudiant est rechargé en RAM et embarque tout ce qui lui est nécessaire : routine d'initialisation, drivers, interpréteurs etc. Ceci permet de choisir librement le langage de programmation et l'environnement, du moment que la chaîne de compilation est capable de produire un exécutable en RAM.

De même, le programme peut être stocker sur une carte SD et téléchargé en RAM par le lanceur résident.

Le programme en RAM à accès à toutes les ressources de la console :

- un peu plus de 3 Mo de RAM externe (le reste étant utilisée par l'écran)
- 320 Ko de RAM interne,
- l'ensemble des périphériques

Seules les mémoires flash interne et externe sont protégées en écriture (par le MPU du microcontrôleur)

De plus, le transfert du programme se fait désormais en passant par la sonde de programmation intégrée à la carte de développement, permettant des vitesses de téléchargement très élevées et, surtout, autorisant le déverminage du code.

1.3 - Prise en main du boîtier

Figure 1: Boîtier vu de face

1 – Écran tactile

5 – Bouton X

2 – Croix directionnelle

3 – Microphones

4 – Bouton RESET

6 – Bouton Y

7 – Bouton B

8 – Bouton A

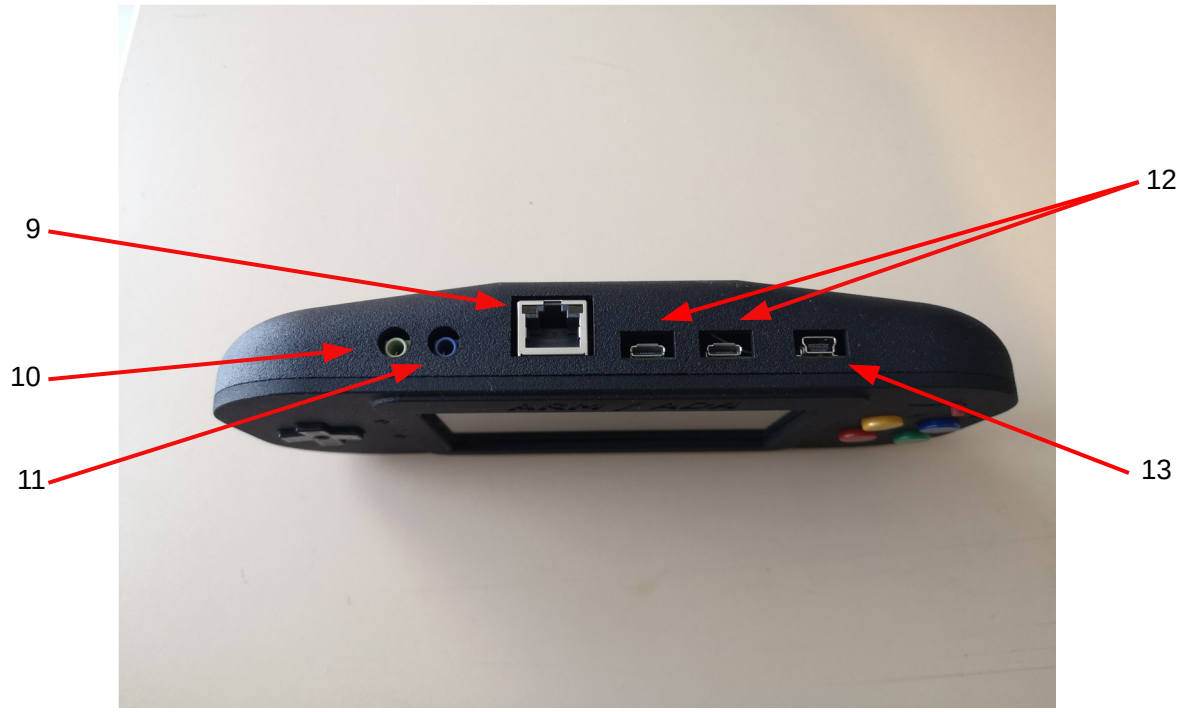

Figure 2: Boîtier vu de dessus

9 – Port Ethernet

10 – Sortie stéréo jack

11 – Entrée stéréo jack

12 – Prises micro USB OTG

13 – Prise mini USB (debug et programmation)

Figure 3: Boîtier vu de dessous

14 – Prise SPDIF

15 – Emplacement carte SD

2 - Organisation des zones mémoires

Plusieurs mémoires (internes ou externes, volatile ou non) sont accessibles dans le boîtier. En fonction du programme s'exécutant (résident en flash ou applicatif de l'étudiant) leurs accès et affectations peuvent changer. Le tableau suivant indique les différentes zone mémoire :

Plage d'adresse	Taille	Type de mémoire	Exécutable	
			Résident	Application
`0x8000000 - 0x8FFFFFF`	1024 Ko	Flash Interne	Code résident (RWX)	Lecture seule (R)
`0x20000000 - 0x2000FFFF`	64 Ko	RAM DTCM	Stack Données partagées (RW) Vecteur d'IT	Stack Données partagées (RW) Vecteur d'IT
`0x20010000 - 0x2004BFFF`	256 Ko	SRAM 1	Données (RW)	Données (RW)
`0x2004C000 - 0x2004FFFF`	16 Ko	SRAM 2	Données (RW)	Données (RW)
`0x90000000 - 0x90FFFFFF`	16 Mo	Flash Externe	Ressources graphiques (R)	Lecture seule (R)
`0xC0000000 - 0xC0340BFF`	3,25 Mo	SRAM Externe	Non utilisé (RWX)	Code application (RWX)
`0xC0340C00 - 0xC03A05FF`	382,5 Ko	SRAM Externe	FrameBuffer 1	FrameBuffer 1
`0xC03A0600 - 0xC03FFFFF`	382,5 Ko	SRAM Externe	FrameBuffer 2	FrameBuffer 2

Tableau 1 : Affectation des zones mémoire

Les zones mémoires communes au résident et à l'applicatif doivent être réinitialisées lorsque l'on passe d'un exécutable à l'autre. L'initialisation des données est à la charge de l'exécutable (applicatif ou résident).

3 - Le programme résident

Le programme résident correspond au programme exécuté après un redémarrage du processeur (reset). Son rôle est d'être un intermédiaire avant le lancement d'une application (développée et testé lors d'une séance de TP, ou se trouvant sur une carte SD) et d'offrir une interface graphique de navigation, ainsi qu'un accès à des tests, notamment pour vérifier le bon fonctionnement du boîtier

Contrairement au modèle précédent, le programme résident n'est pas en charge de la reprogrammation, ni ne fourni de routine d'accès au matériel : il cohabite avec les autres applications et celles-ci doivent fournir toutes les bibliothèques nécessaires à leur fonctionnement.

Le résident s'appuie sur la bibliothèque TouchGFX pour le rendu graphique de l'IHM et utilise la mémoire flash externe pour le stockage des objets graphiques. Les drivers sont eux basés sur la HAL de ST. La programmation de la HAL se fait en C tandis que la bibliothèque TouchGFX est écrite en C++. Du coup, des classes d'encapsulation ont été écrite dans l'application TouchGFX pour accéder aux drivers. Ci dessous, une capture d'écran du résident avec 2 applications disponibles pour être lancées.

Figure 4: Écran type du résident avec 2 applications disponibles

L'arborescence du projet est représenté sur la figure 5 (Arborescence du code du programme résident). On peut notamment distinguer la partie HAL, dont les drivers et la configuration se trouve dans le répertoire « Drivers » et la partie IHM dont la majeur partie est genrée par le générateur d'interface.

Le répertoire GUI contient les classes générées par le générateur d'interface et modifiée par l'utilisateur. Dans ce répertoire se retrouvent :

- Les classes des différents écrans, avec les fonctions callback de gestion des événements (clic, délai, etc.)
- Le conteneur composite LauncherIcon permettant l'affichage des icônes de lancement d'application sur l'interface.

Le répertoire « Helpers » contient les classes qui ne sont pas générées par le

générateur d'interface. Ce sont des classes écrites en dehors du générateur d'interface et facilitant le lien avec le reste du programme. On y trouve:

- les wrappers C++ des drivers HAL nécessaires,
- la classe « launcher » servant à stocker les informations nécessaires au lancement des applications (point d'entrée, type, checksum, zone de stockage du programme, …). Il s'appuie sur la classe LauncherIcon

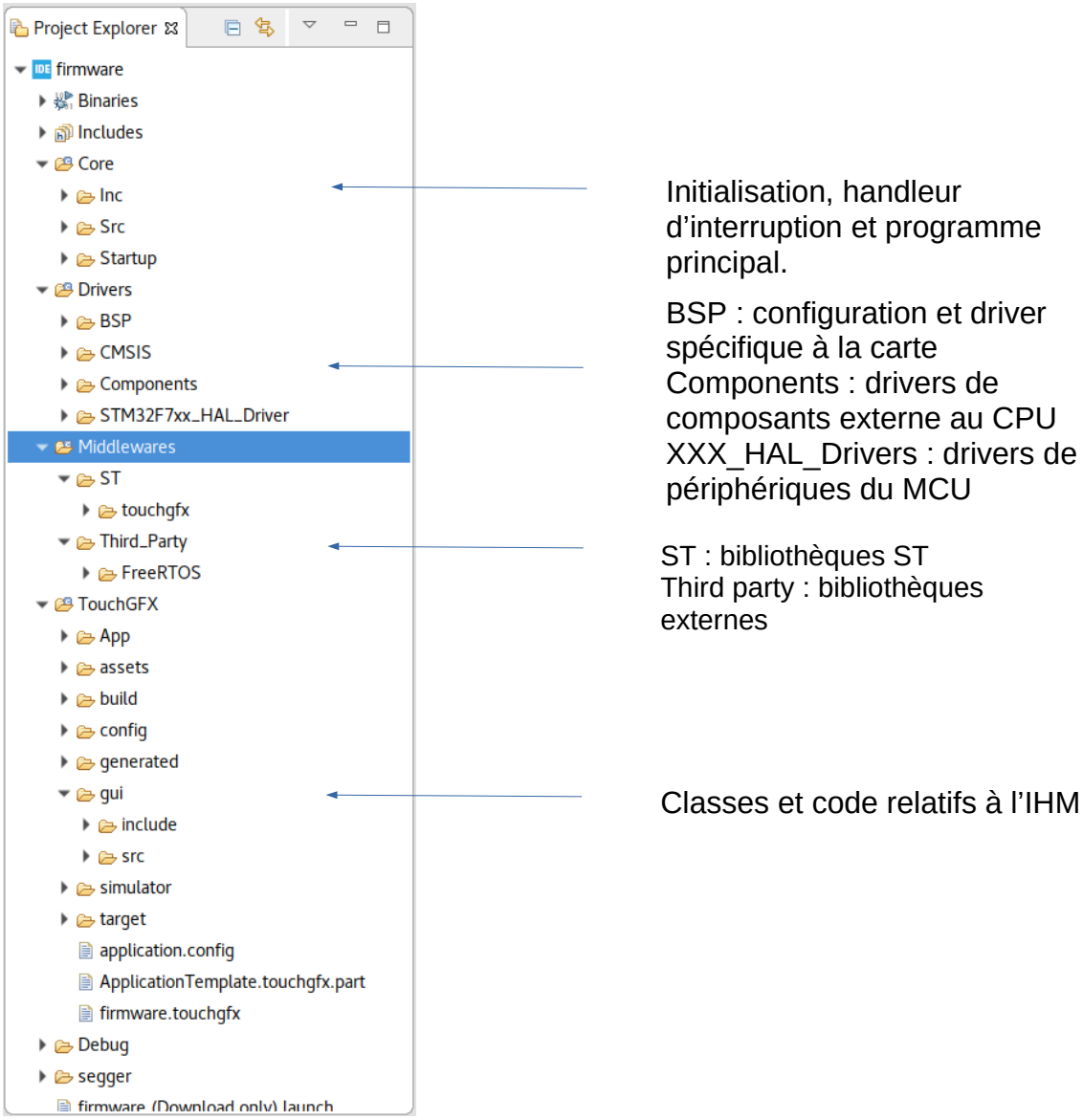

Figure 5: Arborescence du code du programme résident

4 - Les applications

Du fait de la structure logiciel isolant complètement les rôles du résident et des applications, les applications doivent fournir tout ce qui leur est nécessaire pour fonctionner : routines d'initialisation, bibliothèques middleware, ressources graphiques, sonores, etc.

Elles ont accès à tous les périphériques et ressources matériels excepté la flash QSPI et la flash interne, en lecture seule. Le reste est non limité.

4.1 - Application en RAM

De façon à identifier une application présente en RAM (cas d'un code développé par un étudiant dans le cadre d'un TP), un en-tête spécifique doit être présent à l'adresse de début de la RAM externe (0xC0000000). L'en-tête a la structure suivante :

Nom	Type	Description
Signature	String (4 * u8)	Contient le texte « INSA »
Nom de l'application	String (20 * u8)	Contient le nom de l'application avec un zéro terminal (C) limité à 20 caractères.
Descriptif de l'application	String (20 * u8)	Texte libre, pouvant indiquer le type de langage utilisé, le TP, ou autre.
Icône de l'application	u32	Adresse vers une structure de donnée de type image 48x48 pour illustrer l'application sur le lanceur. Si la valeur est null (0x0), l'icône de base est utilisée. Le format de l'icône est décrite dans les annexe.
Point d'entrée de l'application	u32	Adresse du point d'entrée de l'application
Longueur de l'application	u32	Longueur du binaire de l'application (code + data) chargé en RAM en démarrant depuis l'adresse de début de la RAM externe.
Checksum	u32	Somme de contrôle de l'application, depuis l'adresse de début de la SRAM sur l'ensemble de la longueur du binaire. Le calcul du checksum est décrit dans l'annexe

Tableau 2: En-tête de l'application en RAM

4.2 - Débogage d'un programme en RAM

L'une des avancées de cette évolution matérielle est d'utiliser le module de debug (sonde de débogage) intégrée à la carte pour implanter le programme en RAM externe mais aussi permettre son débogage. Le processus pour déboguer un programme est le suivant :

- Le programme est compilé sur le poste de travail de l'étudiant, avec l'environnement de développement qu'il souhaite : le programme est explicitement prévu pour être implanté en RAM.

- Le programme est écrit en RAM par l'outil de debug en passant par la sonde de débogage, un point d'arrêt est mis au début du programme.
- Le boîtier est redémarré, une information de redémarrage à chaud (soft reset) est inscrite en backup RAM. Le résident démarre sans l'animation, et affiche l'icône de l'application détecté en RAM.
- L'utilisateur « lance » le programme en « cliquant » dessus. Le lanceur execute alors le programme en RAM.
- Lorsque le programme atteint le point d'arrêt, le programme s'arrête et l'outil de debug met à jour son affichage (valeur des variables, registres …). Le programme peut alors être débogué normalement.

A cause d'un bug du CPU non corrigé à l'heure actuelle dans le code des sondes ST, la sonde intégrée a été convertie logiciellement en sonde Jlink SEGGER. Il convient donc d'installer sur le poste de travail les drivers Segger pour pouvoir communiquer avec le boîtier. Des scripts sont nécessaires pour écrire le code en RAM.

4.3 - Application sur carte SD

Dans le cas d'une (ou plusieurs) application(s) présente(nt) sur la carte SD, le code n'est pas en RAM externe mais stocké sur une carte SD. De ce fait, les informations sur l'application sont stockées à la racine de la carte SD dans un fichier « apps.yaml ». La structure est la suivante :

```
apps:
    - name:      Nom_de_l_application_1
      descrip:   Description_de_l_application_1
      file:      Chemin_vers_l_application_1
      icon:      Chemin_vers_l_icone_de_l_application_1
      checksum:  Checksum

    - name:      Nom_de_l_application_2
      descrip:   Description_de_l_application_2
      file:      Chemin_vers_l_application_2
      icon:      Chemin_vers_l_icone_de_l_application_2
      checksum:  Checksum
```

Le tableau suivant décrit les différents champs du fichier YAML :

Champs	Type / Taille	Description
name	String (20 * u8)	Nom de l'application, textuelle, tronquée aux 20 premiers caractères.
descrip	String (20 * u8)	Description de l'application, textuelle, tronquée aux 20 premiers caractères.
file	String (100 * u8)	Chemin, à partir de la racine de la carte SD, au format Unix, vers le fichier de l'application.
icon	String (100 * u8)	Chemin, à partir de la racine de la carte SD, au format Unix, vers le fichier contenant l'icône de l'application.

checksum	u32	Somme de contrôle du fichier de l'application, dans sa totalité. Le format est décrit en annexe.

Tableau 3: Description des champs du fichier YAML

Le fichier contenant l'application doit être au format ELF. Lors du chargement, seules les sections localisées en RAM interne et RAM externe sont chargées. Si des sections dépassent ou sont localisées hors de ces espaces, le chargement est arrêté, le système redémarre et un message d'erreur est affiché à l'écran indiquant que le chargement à échoué à cause de sections illégales dans le fichier. Si le fichier ELF à un checksum invalide ou le format du fichier est invalide, un message d'erreur est immédiatement affiché à l'écran avant le chargement du fichier.

4.4 - Exécution d'une l'application

Lors de l'exécution d'une application (préalablement en RAM ou chargée depuis une carte SD), le résident va effectuer une série d'opération visant à passer le contrôle à l'application en RAM. Les opérations suivantes vont être effectuées :

- Vérification de la présence d'un en-tête valide en début de RAM externe.
- Vérification du checksum du programme
- Désactivation des périphériques du microcontrôleur sauf contrôleur d'écran LCD et contrôleur de RAM externes
- Désactivation des interruptions
- Réinitialisation de la stack en DTRAM
- suppression des vecteur d'interruption de la table des vecteur d'IT, située en fin de la DTRAM
- Exécution du programme en SRAM en passant en paramètre la structure d'appel

La structure d'appel est une table contenant un certain nombre d'information utile à l'application et a la structure suivante :

Champs	Type / Taille	Description
version	u32	Version de cette structure
VTOR	u32	Adresse de la table de vecteur d'interruption
returnStruct	u32	Adresse de la structure pour stocker une valeur de retour (u32) et un texte (max 248 caractères), soit 252 octets.
stack	u32	Adresse de début de la stack
frameBuffer	u32	Adresse de début du framebuffer vidéo
paramsString	String (256 * u8)	Chaîne de caractère contenant des paramètres à passer à l'application

Tableau 4: Structure passée à l'exécution

L'adresse de la structure d'appel est stocké dans le registre R0 lors de l'exécution de l'application.

À la fin de l'exécution de l'application, lorsqu'elle rend le contrôle au résident, elle peut indiquer un code retour, lui aussi stocké dans R0. De plus, elle peut écrire un texte dans la chaîne de caractère returnString contenu dans la structure returnStruct dont l'adresse est indiquée dans la structure d'appel, texte qui sera affiché au redémarrage du résident si le code retour ne vaut pas 0. Le résident écrit alors cette valeur de retour dans le champ returnVal de la structure returnStruct.

Nom	Type	Description
returnVal	u32	Valeur de retour de l'application
returnString	String (248 * u8)	Chaîne de caractère produite par l'application à la fin de son exécution.

Tableau 5: Contenu de la structure returnStruct

L'application doit prendre soin de ne pas corrompre ces structures, car ce sont les seuls moyen de communication entre le résident et l'application. Toutes les autres zone mémoire sont libres de modification.

5 - Diagrammes d’activité du résident

Les diagrammes suivants décrivent le comportement du logiciel résident dans différente situations. Il est a noter que chaque situation se focalise sur un scenario (lancement d’une application en RAM, lancement d’une application à partir d’une carte SD, …).

Dans la réalité, ces scenarios coexistent entre eux et, même si une seule application ne peut tourner à la fois, le résident peut être amené à proposer à l’utilisateur de lancer une application en RAM ET de lancer une ou plusieurs application depuis une carte SD, si les deux conditions sont remplies. A l’utilisateur de choisir ce qu’il souhaite lancer.

5.1 - Lancement d’un programme en RAM

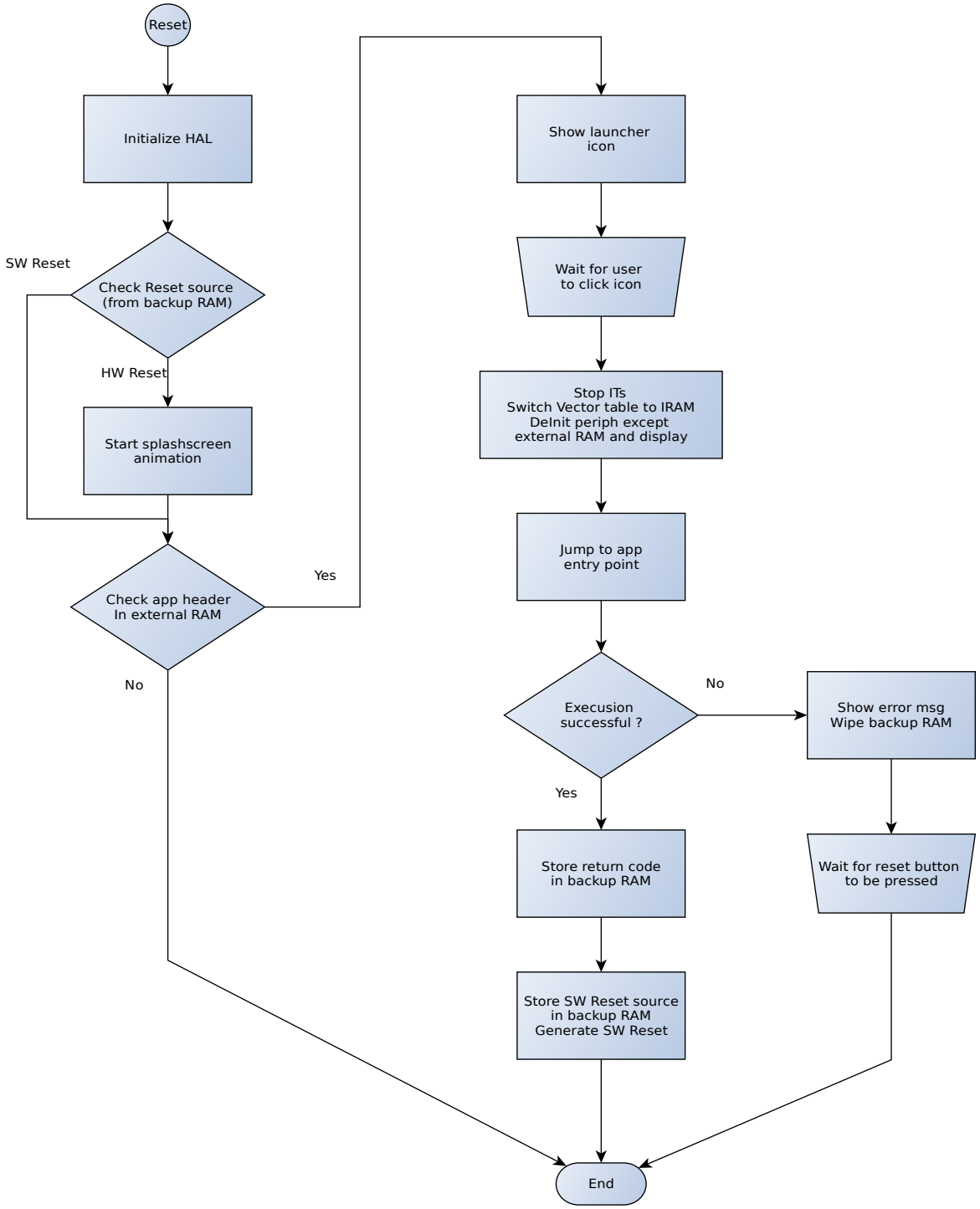

Figure 6: Lancement d'une application présente en RAM

5.2 - Lancement d’une application depuis la carte SD

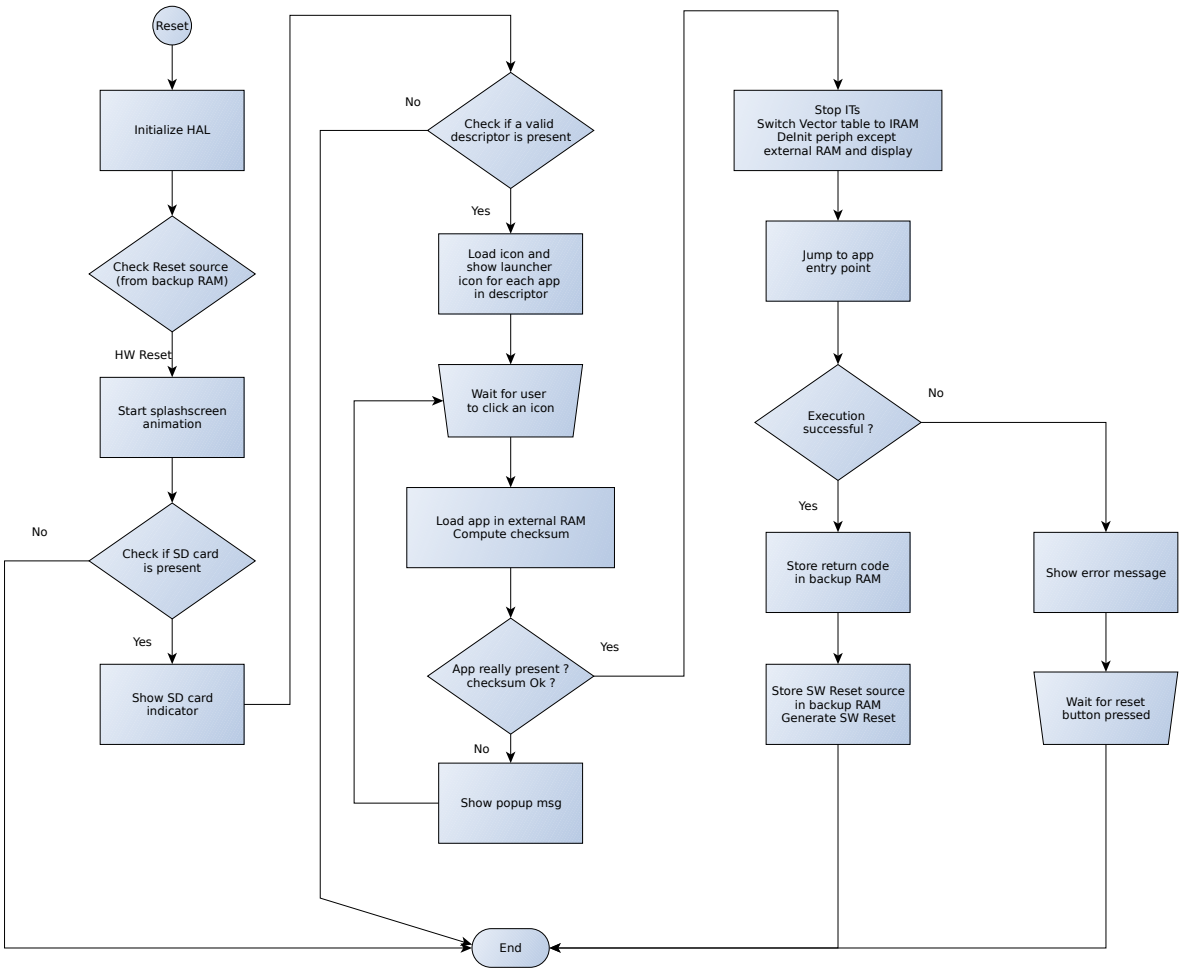

Figure 7: Lancement d'une application depuis la carte SD

5.3 - Lancement des tests

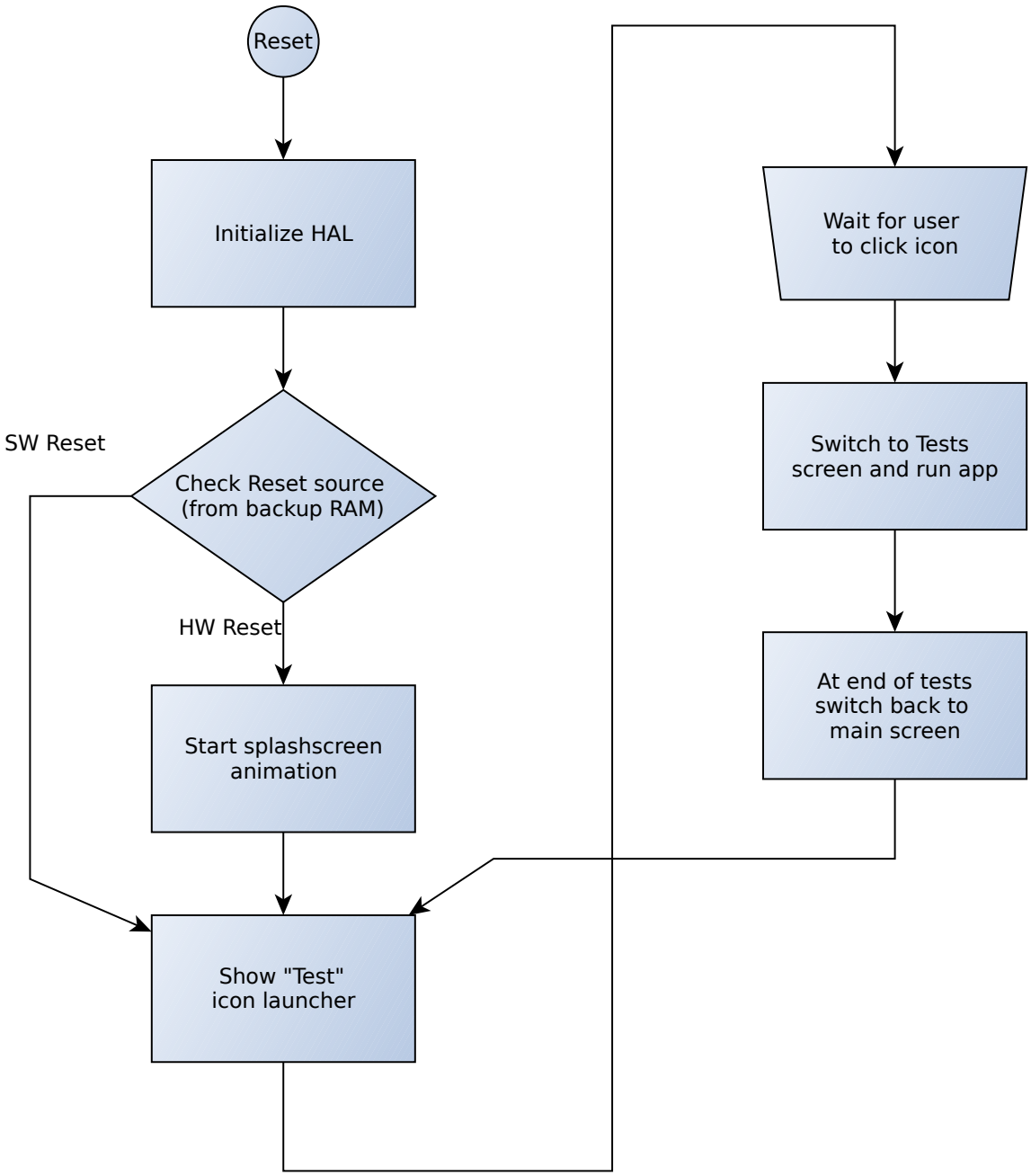

Figure 8: Lancement des tests

6 - Mapping détaillé de la mémoire du résident

Plage d'adresse	Taille	Type de mémoire	Affectation
0x8000000 - 0x80001FF	512 o	Flash Interne (Total 1 Mo)	Vecteur d'IT (R) (en fait seul 456 octets sont utilisés)
0x8000200 - 0x8FFFFFF	1023,5 Ko		Code résident (RX)
0x20000000 - 0x200007FF	2 Ko	RAM DTCM (total 64 Ko)	Stack Système (MSP)
0x20000800 - 0x200008FB	252 o		Données d'échange entre application et résident (RW)
0x200008FC - 0x200008FF	4 o		Type de reset (rstType)
0x20000900 - 0x2000FDFF	61,25 Ko		Heap
0x2000FE00 - 0x2000FFFF	512 o		Vecteur d'IT (Recopie du vecteur en Flash)
0x20010000 - 0x200101DB	276 o	SRAM 1 + SRAM 2 256 Ko + 16 Ko	Structure d'exécution
0x200101DC - 0x2004FFFF	271,73 Ko		Données (RW)
0x90000000 - 0x90FFFFFF	16 Mo	Flash Externe	Ressources graphiques (R)
0xC0000000 - 0xC0340BFF	3,25 Mo	SRAM Externe	Non utilisé (RWX)
0xC0340C00 - 0xC03A05FF	382,5 Ko	SRAM Externe	FrameBuffer 1
0xC03A0600 - 0xC03FFFFF	382,5 Ko	SRAM Externe	FrameBuffer 2

Tableau 6: Mapping mémoire détaillé pour le résident.