

Comment programmer un jeu pour la console Vtech V.Smile

Adrien Destugues – PulkoMandy
Capitole du Libre 2022

Comment programmer un jeu pour la console Vtech V.Smile

Historique

Présentation du matériel

Outils de développement

Historique

Video Technology Limited, créée à Hong Kong en 1976

Fabricant de jeux électroniques portables de type « Game&Watch »

1980 : premier jeu électronique éducatif : Lesson One



1982 : Console Vtech CreatiVision

1983 : Ordinateurs Vtech Laser

Laser 128 : Clone de l'Apple][avec une ROM réécrite, inattaquable par Apple

1985 : Entrée sur le marché des téléphones DECT

1989 : Precomputer 1000, ordinateur éducatif

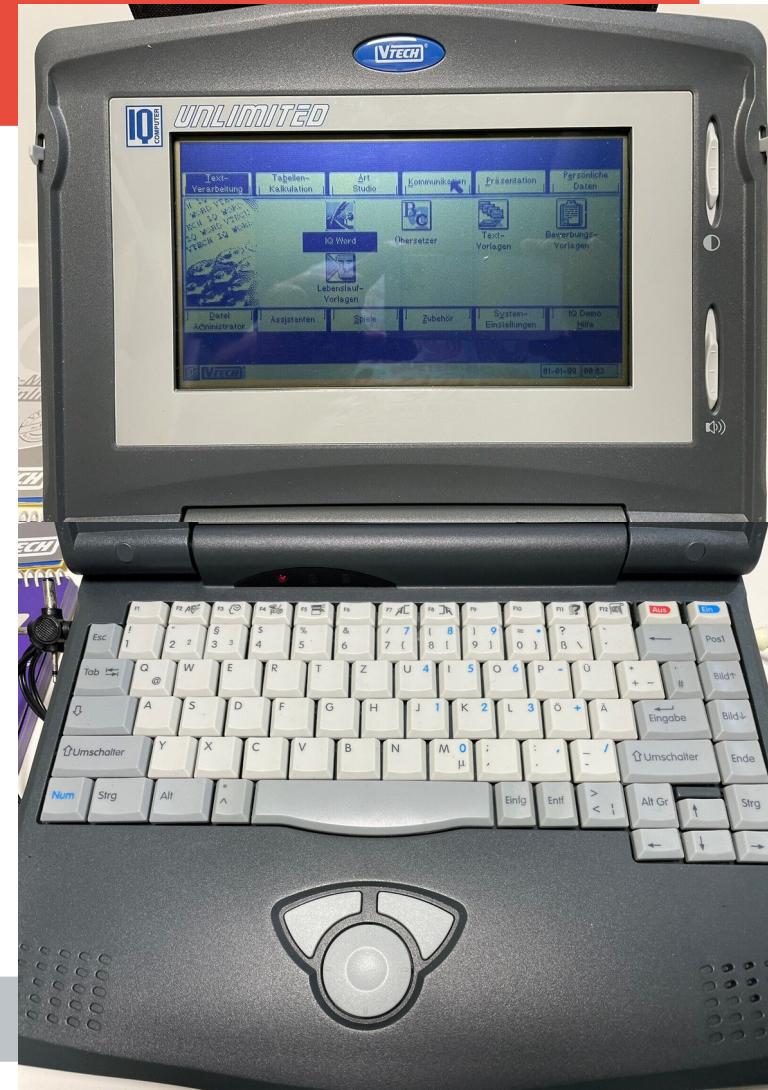


Années 1990 : abandon du marché des ordinateurs Laser et compatibles PC

1997 : ordinateur portable éducatif VTech Manager

Objectif : un ordinateur pas cher pour les collégiens pour lutter contre la fracture numérique

Processeur Motorola ColdFire (comme les PDA Palm)



2000 : le PDA VTech Helio

Dernière tentative de retour sur le marché non éducatif

Processeur MIPS à 75MHz, 8Mo de RAM

Le code source de l'OS et des applications est publié

(mais pas libre : interdiction de l'utiliser sur un autre matériel)



La console V.Smile



Première version en 2004

V.Smile Pocket en 2005

V.Smile Baby en 2006 (pas compatible)

V.Smile Motion et V.Smile Cyber Pocket en 2008

Fin de la production en 2010

Remplacée en 2007 par la V.Smile Pro

(console de salon)

Remplacée en 2010 par la MobiGo

(console portable)

Jeux



Partenariats avec Disney, DC comics, ...

Jeux éducatifs

Graphismes proches de la
MegaDrive

Graphismes identiques pour certains
jeux



Accessoires



Reverse engineering

Exploration du matériel démarrée par Bushing et Segher du groupe HackMii

Au départ pour la console JungleTac Vii qui utilise le même processeur

Écriture de l'émulateur « unununium » (2008-2010)

Certains jeux fonctionnent

Pas de son

Ensuite, intégration de l'émulation dans MAME par MooglyGuy

Ajout du son en 2018

Bonne compatibilité avec la plupart des jeux aujourd'hui

Autres consoles supportées : Jakks Pacific, Mobigo, ...



Reverse engineering

Exploration de la V.Smile par BMX (vers 2009)

Démontage de la console

Dump de cartouches

Découverte que le processeur utilisé est le même que dans la Vii

Mise en commun du travail

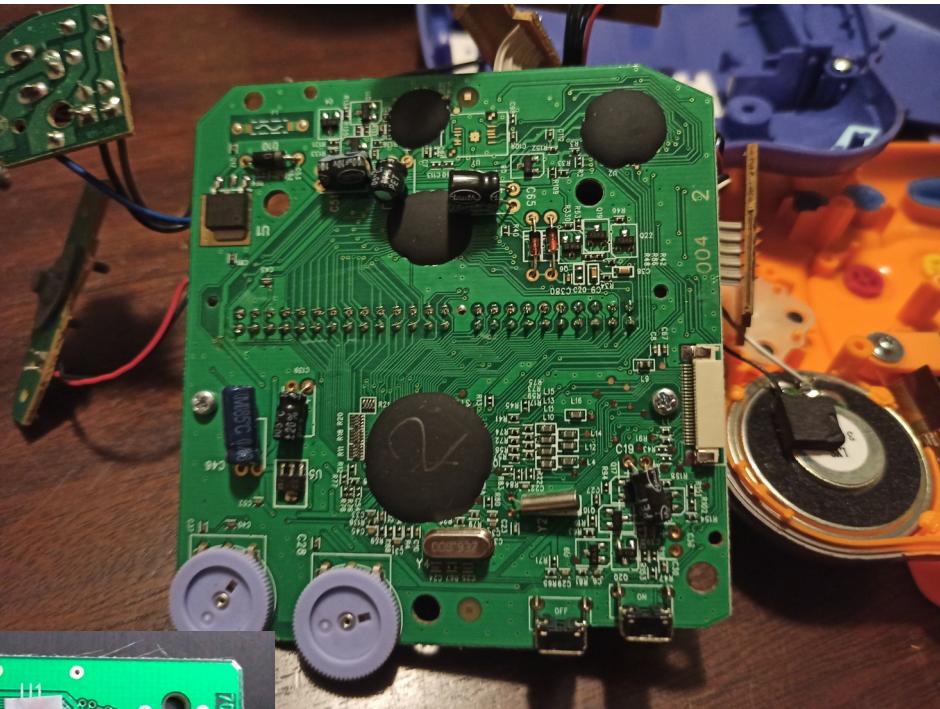
Par où commencer ?

La carte mère de la console ne comporte que des « blobs »

Composants sans package, déposés directement sur la carte mère et protégés par une goutte d'epoxy

Pas de marquage identifiable

Même chose pour les cartouches



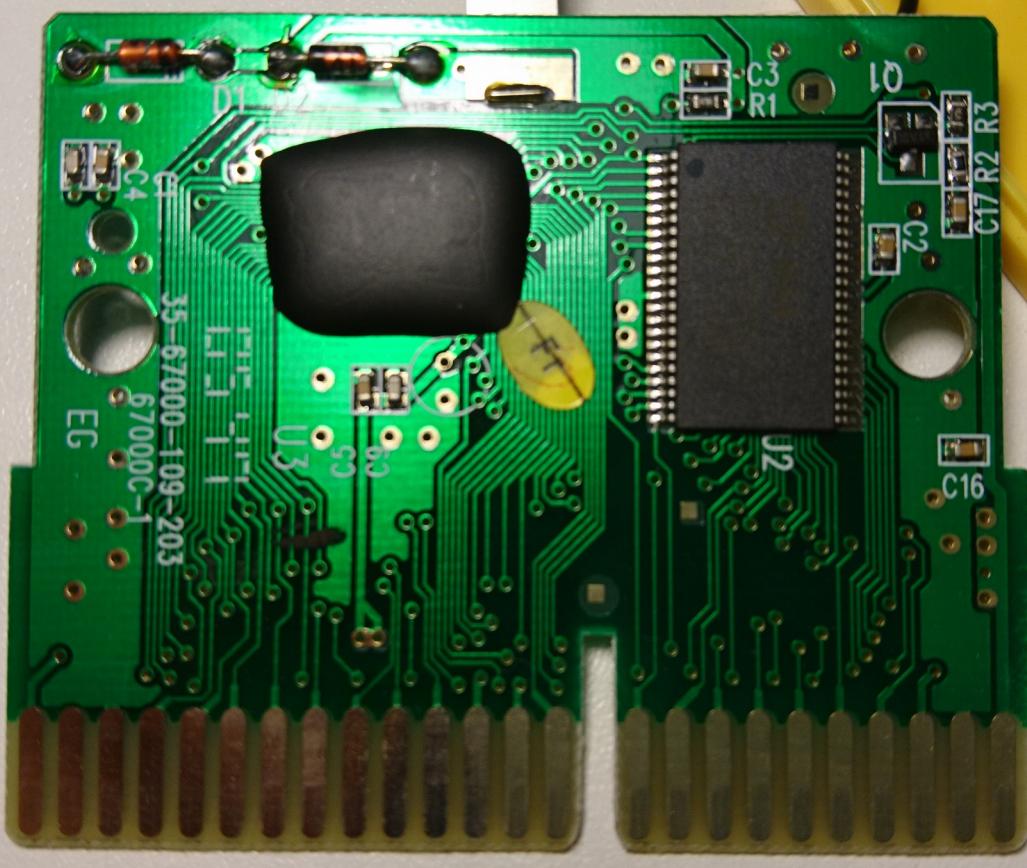
Par où commencer ?

... presque toutes les cartouches !

La cartouche « Art Studio » intègre de la RAM supplémentaire et c'est un composant standard

On peut en déduire le brochage du port cartouche

Ce qui permet de lire une cartouche pour en extraire le code et les données



Pin N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Front	/WE	/OE	GND	D3	D4	D5	D6	D11	D12	D13	D15	A0	A1	A3	x	A5	A7	A18	A9	A11	A13	A15	A20	GND	Card detect
Back	VDD	RAM CSB	Sense	ROM CSB2	D2	D1	D0	D7	D10	D9	D8	D14	A16	A2	x	A4	A6	A17	A8	A10	A12	A14	A19	A21	ROM CSB1

Qu'est-ce qu'on trouve dedans ?

Première analyse : chercher les chaînes de caractères

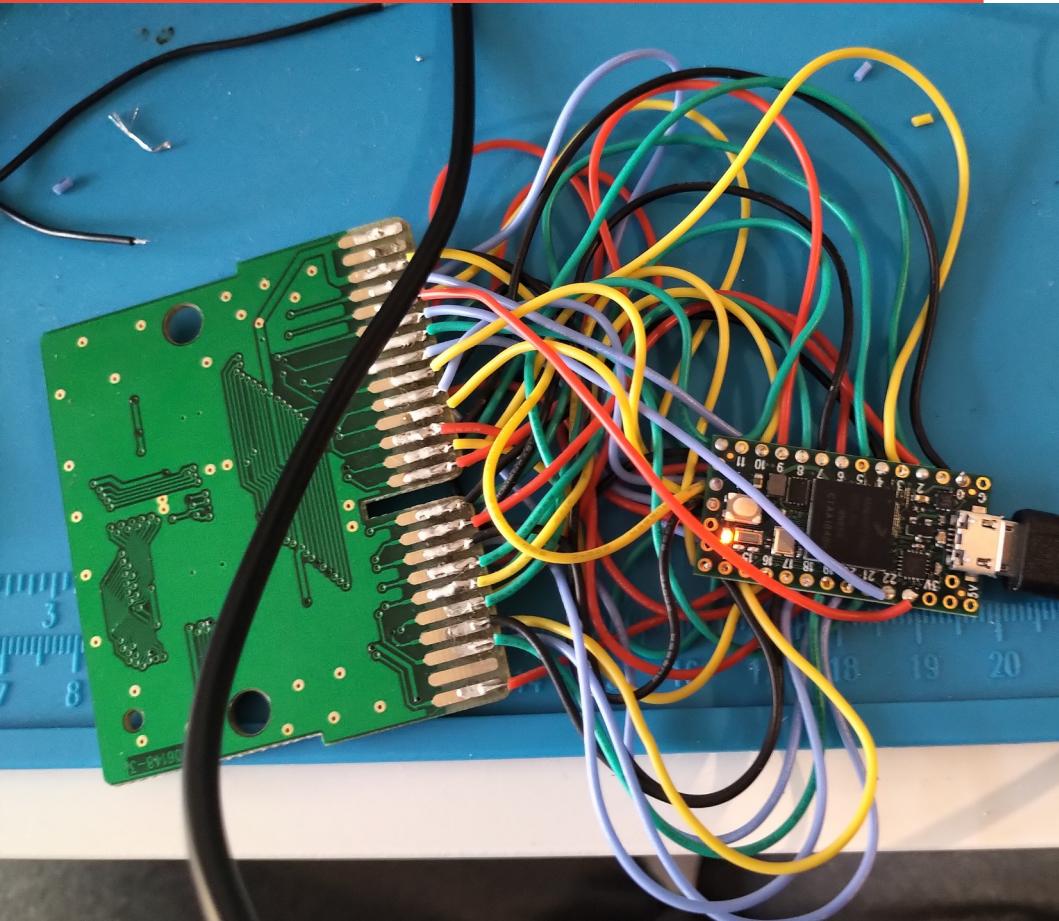
```
strings -n 20
```

```
ver: «u'nSPIDE ver 1.6.2»
```

```
user: «sunplus»
```

```
body: «SPG243»
```

On obtient le nom du fabricant, du composant utilisé et de l'architecture CPU !



Et le reste du matériel ?

Connaître l'architecture CPU ne suffit pas

Comment fonctionne la sortie vidéo ? Le son ? Les manettes ?

On ne dispose pas de documentation sur le composant exact utilisé par la V.Smile

Trois approches pour en savoir plus :

Désassemblage et étude du code des jeux

Écriture d'un émulateur

Expérimentations sur le matériel en écrivant du code de test

Approches à combiner entre elles

Une cartouche reprogrammable

V.kart : une cartouche avec une mémoire flash

Permet de lancer du code sur la vraie console

Comparer l'exécution avec l'émulateur sur du code simple

Mesurer les timings, etc

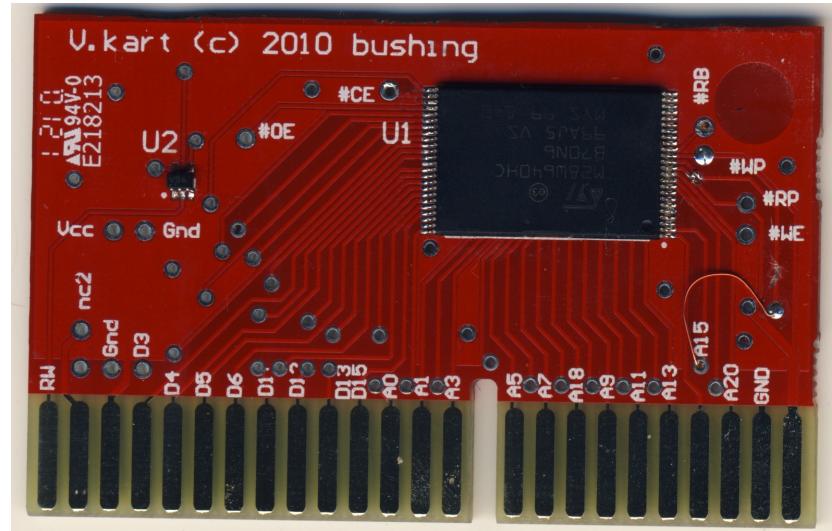
Processus de développement assez pénible

Assembler le code (ou l'écrire directement en binaire)

L'écrire sur la cartouche

Mettre la cartouche dans la console pour tester

Recommencer...



Premiers outils de développement

Un interpréteur Forth

Un langage interprétré très simple

Très peu de code à écrire pour l'interpréter

Beaucoup plus confortable que l'assembleur

Surtout sur un CPU qu'on connaît encore mal

Possibilité eventuelle de faire un bootloader pour charger facilement du code

Mais ça n'aboutira pas

Premiers outils de développement

u'nSP IDE

IDE fourni par SunPlus/GeneralPlus

Basé sur GCC 2.93

Code source publié mais pas compilable, et il manque les sources de l'assembleur

Ne respecte pas la licence GPL

Permet malgré tout d'écrire du code pour tester la console

Et ensuite ?

Le plus dur est fait !

Un émulateur fonctionnel

Une cartouche programmable

Divers outils (assembleur, désassembleur) pour faciliter le développement

... mais cela coïncide avec la fin de vie de la console (2010)

Perte d'intérêt des développeurs d'émulateurs une fois la plupart des jeux fonctionnels

Peu d'intérêt des potentiels développeurs de jeux

Sur unununium

Patchs de RebeccaRGB pour corriger plusieurs bugs (2017-2018)

Sur MAME

Patchs de MooglyGuy pour améliorer l'émulation à partir du code de unununium, ajouter le son, etc

Patchs de plusieurs contributeurs pour améliorer l'émulation, ajouter la V.Smile Baby et d'autres consoles

2020 : création d'un serveur Discord pour rassembler à nouveau les personnes intéressées

Autres sources de documentation

Datasheets et manuels des composants

SunPlus/GeneralPlus fournit de la documentation pour certains composants

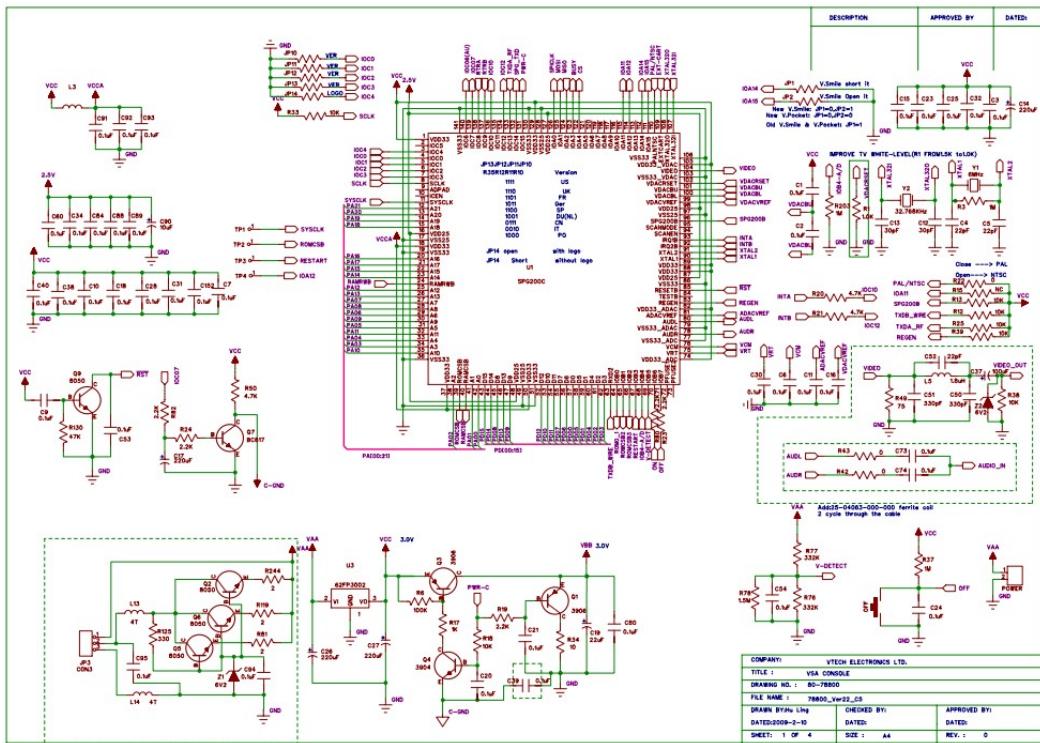
Par exemple dans des packages destinés aux universités avec une carte d'évaluation

Aussi des versions qui ont fuité sur internet

Schémas de la console

Schémas de la V.Smile Motion publiés par la FCC suite à sa certification

Malgré la demande de VTech de les garder confidentiels



Présentation du matériel

SunPlus / GeneralPlus u'nSP



Architecture 16 bits vaguement inspirée du DEC PDP-11 (1970)

8 Registres :

ID	Name	Function
0	SP	Stack Pointer
1	R1	General Purpose
2	R2	General Purpose
3	R3	General Purpose
4	R4	General Purpose
5	BP	Base Pointer
6	SR	Status Register
7	PC	Program Counter



Bits	Name	Description
0-5	CS	Code Segment
6	C	Carry Flag
7	S	Sign Flag
8	Z	Zero Flag
9	N	Negative Flag
10-15	DS	Data Segment

Encodage des instructions sur 16 bits assez régulier :

Bits	15-12	11-9	8-6	5-3	2-0
Contents	Opcode0	Operand A	Opcode1	OPN	Operand B
Contents	Opcode0	Operand A	Opcode1	6-bit Immediate	

SunPlus / GeneralPlus u'nSP

Adressage par mots de 16 bit uniquement

Impossible d'adresser un seul octet

Mémoire segmentée

64 segments de 64 Ki-mots de 16 bits, soit un total de 4Mio

CS et DS pour choisir les segments utilisés (code et données)

Stockés dans le SR (status register)

Incrémentés automatiquement si nécessaire

Instruction CALL pour appeler une routine dans un autre segment

Modifie et restaure automatiquement la valeur dans SR

Opérations arithmétiques, modes d'adressage

Instruction	Opcode0	Operand A	Opcode1	OPN	Operand B
ADD	0	Op. A	Addr Mode	Param	Op. B
ADC	1	Op. A	Addr Mode	Param	Op. B
SUB	2	Op. A	Addr Mode	Param	Op. B
SBC	3	Op. A	Addr Mode	Param	Op. B
CMP	4	Op. A	Addr Mode	Param	Op. B
NEG	6	Op. A	Addr Mode	Param	Op. B
XOR	8	Op. A	Addr Mode	Param	Op. B
LD	9	Op. A	Addr Mode	Param	Op. B
POP	9	First reg - 1	2	Register count	Stack pointer reg
RETF	9	5 (SR)	2	2 (SR, PC)	0 (SP)
RETI	9	5 (SR)	2	3 (SR, PC, FR)	0 (SP)
OR	A	Op. A	Addr Mode	Param	Op. B
AND	B	Op. A	Addr Mode	Param	Op. B
TEST	C	Op. A	Addr Mode	Param	Op. B
ST	D	Op. A	Addr Mode	Param	Op. B
PUSH	D	Last reg -1	2	Register count	Stack pointer reg
	F	Reserved for special instructions			

Addressing mode	Opcode0	Operand A	Opcode1	OPN	Operand B
[BP+Imm6]	ALU op	Op. A	0	6-bit immediate	
#Imm6	ALU op	Op. A	1	6-bit immediate	
Special (POP, PUSH)			2		
[Rs]	ALU op	Op. A	3	0	Rs
[Rs--]	ALU op	Op. A	3	1	Rs
[Rs++]	ALU op	Op. A	3	2	Rs
[++Rs]	ALU op	Op. A	3	3	Rs
D:[Rs]	ALU op	Op. A	3	4	Rs
D:[Rs--]	ALU op	Op. A	3	5	Rs
D:[Rs++]	ALU op	Op. A	3	6	Rs
D:[++Rs]	ALU op	Op. A	3	7	Rs
Rs	ALU op	Op. A	4	0	Rs
#Imm16	ALU op	Op. A	4	1	Rs
From [Addr16]	ALU op	Op. A	4	2	Rs
To [Addr16]	ALU op	Op. A	4	3	Rs
Rs ASR shift	ALU op	Op. A	4	4 + (shift - 1)	Rs
Rs LSL shift	ALU op	Op. A	5	shift - 1	Rs
Rs LSR shift	ALU op	Op. A	5	4 + (shift - 1)	Rs
Rs ROL shift	ALU op	Op. A	6	shift - 1	Rs
Rs ROR shift	ALU op	Op. A	6	4 + (shift - 1)	Rs
[Addr6]	ALU op	Op. A	7	6-bit address	

Sauts et interruptions

Instruction	Opcode0	Operand A	Opcode1	6-bit Immediate	
JB, JNAE, JCC	0	7	Direction	Jump offset	
JAE, JNB, JCS	1	7	Direction	Jump offset	
JGE, JNL, JSC	2	7	Direction	Jump offset	
JL, JNGE, JSS	3	7	Direction	Jump offset	
JNE, JNZ	4	7	Direction	Jump offset	
JE, JZ	5	7	Direction	Jump offset	
JPL	6	7	Direction	Jump offset	
JMI	7	7	Direction	Jump offset	
JBE, JNA	8	7	Direction	Jump offset	
JA, JNBE	9	7	Direction	Jump offset	
JLE, JNG	A	7	Direction	Jump offset	
JG, JNLE	B	7	Direction	Jump offset	
JVC	C	7	Direction	Jump offset	
JVS	D	7	Direction	Jump offset	
JMP	E	7	Direction	Jump offset	
	F	7	Reserved for special instructions		

Instruction	Opcode0	Operand A	Opcode1	6-bit Immediate	Second word
CALL	F	?	1	CS: value	PC value
GOTO	F	7	2	CS: value	PC value
Instruction	Opcode0	Operand A	Opcode1	6-bit Immediate	
INT OFF	F	?	5	00	
INT IRQ	F	?	5	01	
INT FIQ	F	?	5	02	
INT IRQ,FIQ	F	?	5	03	
FIR_MOV ON	F	?	5	04	
FIR_MOV OFF	F	?	5	05	
IRQ OFF	F	?	5	08	
IRQ ON	F	?	5	09	
FIQ OFF	F	?	5	0C	
FIQ ON	F	?	5	0E	
BREAK	F	?	5	20	

Multiplications

Instruction	Opcode0	Operand A	Opcode1	OPN	Operand B
MAC (unsigned*signed)	F	Op.A	2 + (N ≥ 8)	N & 7	Op.B
MAC (signed*signed)	F	Op.A	6 + (N ≥ 8)	N & 7	Op.B
MUL (unsigned*signed)	F	Op.A	0	1	Op.B
MUL (signed*signed)	F	Op.A	4	1	Op.B

L'opération MAC (Multiply-Accumulate) multiplie deux tables de jusqu'à 16 valeurs et additionne tous les résultats

Utilisé en traitement du signal numérique

Résultat stocké dans R3 et R4

Organisation de la mémoire

Start	End	Description
0x000000	0x0027FF	RAM (20Ko - 10Kwords)
0x002800	0x002FFF	PPU (Picture Processing Unit) – contrôleur graphique
0x003000	0x0037FF	Son
0x003D00	0x003DFF	Entrées-Sorties (GPIO) et autres périphériques (UART, ...)
0x003E00	0x003E03	DMA (Direct Memory Access)
0x004000	0x3FFFFF	Mémoire externe (cartouche)

Picture Processing Unit

Résolution fixe : 320 x 240 pixels

2 calques fonctionnant avec des « tiles » + jusqu'à 256 sprites

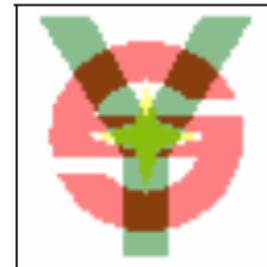
Effet de transparence (4 niveaux)

Palette de 256 couleurs configurables parmi 32768

+ effet d'assombrissement global de l'écran

Effet de déformation verticale et horizontale des calques

Canal DMA dédié pour copier des données de la cartouche vers la RAM



Picture Processing Unit : calques

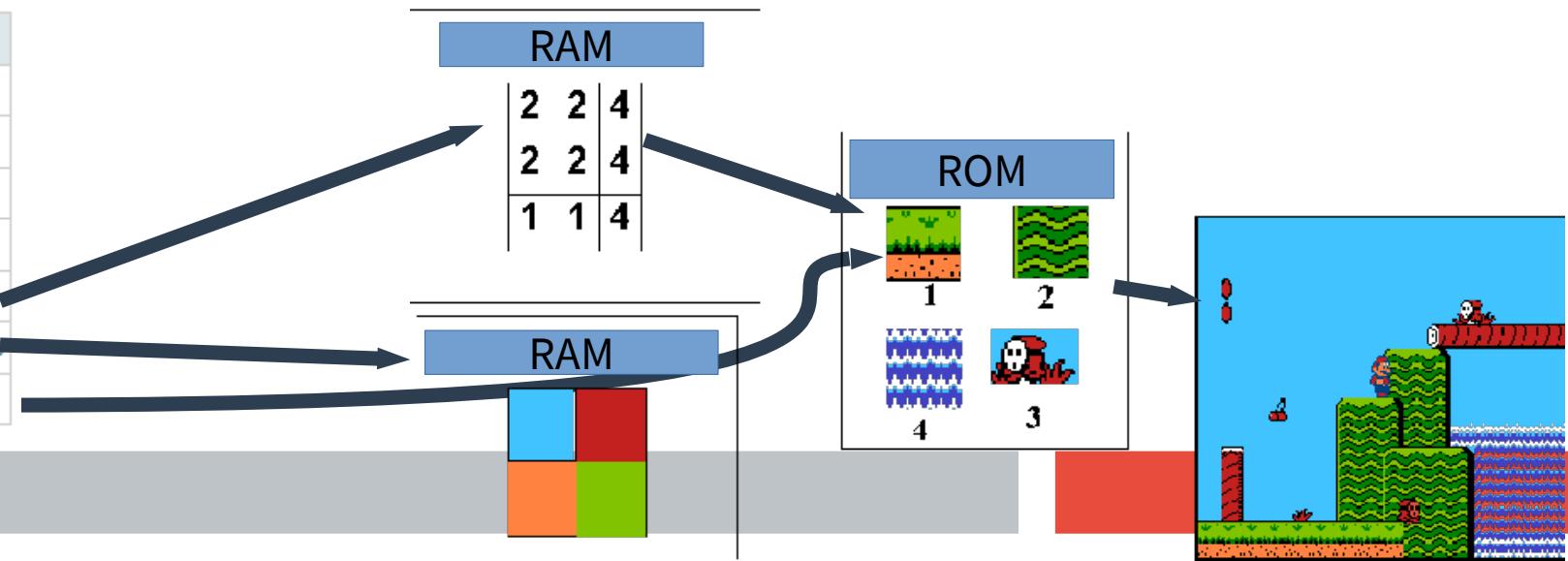
Chaque calque a 4, 16, 64 ou 256 couleurs

Les tuiles font entre 8x8 et 64x64 pixels

Possibilité de miroir vertical et horizontal sur le contenu des tuiles

Défilement vertical et horizontal, déformation verticale/horizontale

Address	Function
2810/2816	X position
2811/2817	Y position
2812/2818	Attributes
2813/2819	Control
2814/281A	Tilemap address
2815/281B	Attribute map address
2820/2821	Tile data segment



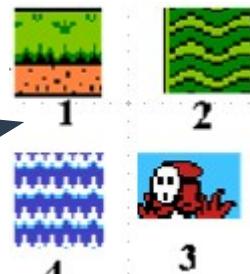
Picture Processing Unit : sprites

Éléments mobiles visibles par-dessus (ou par-dessous) les calques

Chaque sprite contient 1 tuile

Attributs similaires à ceux des tuiles dans les calques

0	Sprite char select
1	Sprite X position
2	Sprite Y position
3	Sprite attributes



- Bit depth (bits 0 and 1)
 - 00 - 2 bits per pixel
 - 01 - 4 bits per pixel
 - 10 - 6 bits per pixel
 - 11 - 8 bits per pixel
- Bit 2: Mirror the tiles horizontally
- Bit 3: Mirror the tiles vertically
- Bits 4-5: Width of tiles
 - 00 - 8 pixels
 - 01 - 16 pixels
 - 10 - 32 pixels
 - 11 - 64 pixels
- Bits 6-7: Height of tiles
 - Same format as width register
- Bits 8-11: Palette bank to use
- Bits 12-13: Layer depth (whether it is under or over sprites and the other layer)
 - 00 - bottom layer
 - 01 - second-bottom layer
 - 10 - second-top layer
 - 11 - top layer

Son

16 canaux PCM ou ADPCM

Fréquence et volume configurable
(séparément pour chaque canal)

Contrôle automatique d'enveloppe de volume

Address	Function
30×0	Channel X sample address
30×1	Channel X control (bit 14: 16M, 13-12: ToneMode, 11-6: Loop address segment, 5-0: sample address segment)
30×2	Channel X loop address
30×3	Channel X panning and volume (7 bits each)
30×4	Channel X envelope target volume and increment
30×5	Channel X envelope CNT and EDD (?)
30×6	Channel X repeat count (bits 15-9), repeat enable (8), Envelope load (7-0)
30×7	Channel X IRQ address (15-9), IRQ enable, envelope segment (6-0)
30×8	Channel X envelope address
30×9	Channel X wave data 0
30xA	Channel X ramp down offset (15-9) and envelope address offset (8-0)
30xB	Channel X wave data
32×0	Phase high bits (2-0)
32×1	Phase accumulator high (2-0)
32×2	Target phase high (2-0)
32×3	Ramp down clock

Address	Function
3400	Channel enable (1 bit per channel)
3401	Main volume (7 bits)
3402	Channel FIQ enable
3403	Channel FIQ status
3404	Beat base counter
3405	Beat counter (13-0), IRQ enable (15) and status (14)
3406	Envelope interval, channels 0-3
3407	... channels 4-7
3408	... channels 8-11
3409	... channels 12-15
340A	Envelope fast ramp-down
340B	Channel stop status (1 bit per channel)
340C	Zero cross enable (1 bit per channel)
340D	Control flags: NoInt (9), LPEn (8), HVol(7-6), SOF (5), Init (3), PCM (2)
340F	Channels status
3410	Left mixer input
3411	Right mixer input
3412	Left mixer output
3413	Right mixer output
3414	Channel repeat enable (1 bit per channel)
3415	Channel envelope mode
3416	Channel tone release control
3417	Channel envelope IRQ status
3418	Channel pitch band enable

Contrôleurs de jeu

Les deux ports pour les contrôleurs sont reliés à une seule UART

Contrôle de flux matériel pour adresser chaque port séparément

Le contrôleur signale quand il veut envoyer un octet (« Request to send »)

La console répond quand elle est prête (« Clear to send »)

Le contrôleur peut ensuite utiliser l'UART (Rx/Tx)

Joystick analogique (8 positions possibles sur chaque axe)

Possibilité de contrôler les LEDs des boutons des manettes

Envoi régulier d'un octet « keep alive » (permet de détecter une manette débranchée)

Somme de contrôle sur les données reçues

Algorithme pas encore complètement documenté, à faire à partir du désassemblage du BIOS de la console

V.Link

Il s'agit d'un port SPI ou I2C (à confirmer)

Accès direct à une EEPROM dans le V.Link

Possibilité de connecter d'autres périphériques sur les consoles disposant de ce port

Pour les autres il faudra se contenter des ports pour les contrôleurs

Outils de développement

Émulateur et debugger

u'nSP IDE

Émulateur très simple du CPU uniquement, atteint vite ses limites

unununium, V.Frown

Émulateurs spécifiques pour la V.Smile

Pas de debugger

MAME

Projet émulant de nombreuses machines

Intègre un débugger austère mais efficace

The screenshot shows the MAME debugger interface with several panes:

- Registers:** Shows CPU registers (PC, SP, ISP, USP) and memory pointers (D0-D7, A0-A7, PREF_ADDR, PREF_DATA) with their current values.
- Memory Dump:** A large pane showing memory starting at address 000926. A specific instruction at address 000926 is highlighted in yellow: `move.w #$7080, $400000.1`. The right side of this pane is labeled **MACHINE CODE**.
- COMMAND LINE & HISTORY:** A bottom pane containing the text:

```
MAME new debugger version 0.139[RR] (Aug 8 2010)
Currently targeting vsav (Vampire Savior: The Lord of Vampire (Euro 970519))
```

Émulateur et debugger

On pourrait aller beaucoup plus loin, mais c'est du travail !

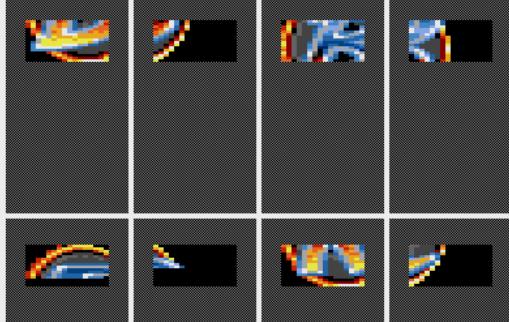
ACE - Analyseur sprites hard

Palette des sprites hard

Contenu des sprites hard

Encre 1 : ASIC #800
 Encre 2 : ASIC #C02
 Encre 3 : ASIC #E26;
 Encre 4 : ASIC #E2A.
 Encre 5 : ASIC #E4E
 Encre 6 : ASIC #062;
 Encre 7 : ASIC #084
 Encre 8 : ASIC #2A6
 Encre 9 : ASIC #4C8;
 Encre 10 : ASIC #8EA
 Encre 11 : ASIC #000
 Encre 12 : ASIC #444;
 Encre 13 : ASIC #666
 Encre 14 : ASIC #AAA
 Encre 15 : ASIC #EEE;

Couleur de fond : Afficher les sprites éteints



ACE - Analyseur DMA

DMA 0	DMA 1	DMA 2
<input checked="" type="radio"/> État : Arrêté	<input checked="" type="radio"/> État : Arrêté	<input type="radio"/> État : En exécution
Adresse : &CD2	Adresse : &0000	Adresse : &00FA
Unité de pause : 0 (0:0)	Unité de pause : 0 (0:0)	Unité de pause : 0 (0:16)
Bouclage en : &0000 (0)	Bouclage en : &0000 (0)	Bouclage en : &00F4 (4008)
04 09 LOAD R9, &04	00 00 LOAD R0, &00	08 08 LOAD R8, &08
00 09 LOAD R9, &00	00 00 LOAD R0, &00	15 10 PAUSE &015
20 40 STOP	00 00 LOAD R0, &00	04 08 LOAD R8, &04
20 40 STOP	B8 12 PAUSE &2B8	14 10 PAUSE &014
0B 08 LOAD R8, &0B	00 00 LOAD R0, &00	01 40 LOOP
0A 08 LOAD R8, &0A	00 00 LOAD R0, &00	20 40 STOP
03 10 PAUSE &003	00 00 LOAD R0, &00	00 09 LOAD R9, &00
0B 08 LOAD R8, &0B	00 00 LOAD R0, &00	08 10 PAUSE &008
05 10 PAUSE &005	00 00 LOAD R0, &00	08 09 LOAD R9, &08
0A 08 LOAD R8, &0A	00 00 LOAD R0, &00	08 09 LOAD R9, &08

ACE - Éditeur Z80

Flags

P	Z	H	PE	NN	NC
---	---	---	----	----	----

Registres

AF : & 00 54	AF' : & 80 90
BC : & F6 80	BC' : & F4 80
DE : & D5 7B	DE' : & 07 70
HL : & CF 7B	HL' : & F1 6C
IX : & D5 6C	SP : & BF E6
IY : & 00 F2	PC : & D1 D2
I : & BE	R : & 1D

Interruptions

État : <input checked="" type="radio"/> Désactivées
Mode : <input checked="" type="radio"/> IM 1

Pile

-2:&DC86	W : k	57 7F 6B
-1:&A196	...	00 00 00
0:&D008	...	G CD 47
1:&830C	...	FD E1 DD
2:&22F3	...	08 01 10
3:&21F3	...	10 0A 08
4:&082D	...	10 08 09
5:&761B	...	BB B8 BA
6:&2020	...	0C 08 01

ACE - Gestionnaire de symboles

Motif de filtrage : #?

Nom	Adresse	Contenu désassemblé	Contenu hexadécimal	Co
KM_GET_JOYSTICK	&BB24	RST 1,&9DE5	CF E5 9D CF D8 9E CF C4	...
KM_GET_REPEAT	&BB3C	RST 1,&9E2F	CF 2F 9E CF F6 9D CF F2	...
KM_GET_SHIFT	&BB30	RST 1,&9EC9	CF C9 9E CF E2 9E CF CE	...
KM_GET_STATE	&BB21	RST 1,&9D38	CF 38 9D CF E5 9D CF D8	...
KM_GET_TRANSLATE	&BB2A	RST 1,&9EC4	CF C4 9E CF DD 9E CF C9	...
KM_INITIALISE	&BB00	RST 1,&9B5C	CF 5C 9B CF 98 9B CF BF	...
KM_READ_CHAR	&BB09	RST 1,&9BC5	CF C5 9B CF FA 9B CF 46	...
KM_READ_KEY	&BB1B	RST 1,&9CE1	CF E1 9C CF 45 9E CF 38	...
KM_RESET	&BB03	RST 1,&9B98	CF 98 9B CF BF 9B CF C5	...
KM_SET_CONTROL	&BB33	RST 1,&9E2	CF E2 9E CF CE 9E CF 34	...
KM_SET_DELAY	&BB3F	RST 1,&9DF6	CF F6 9D CF F2 9D CF FA	...
KM_SET_EXPAND	&BB0F	RST 1,&9C46	CF 46 9C CF B3 9C CF 04	...
KM_SET_LOCKS	&BD3A	RST 1,&9D3C	CF 3C 9D CF FE 9B CF 60	...

Effacer tout

Symbole

Nom :

Adresse : & 00 00

Réglages avancés

Conserver les ROMs d'extension quand une cartouche est insérée

Firmware :

- 3 - CPC6128 - clavier français
- 11 - CPC6128 - clavier français
- 0.5 - Amstrad CPC CP/M ROM

ROMs d'extension

0 à 127	128 à 255		
0 à 31	32 à 63	64 à 95	96 à 127
0 à 7	8 à 15	16 à 23	24 à 31

ROM 0 : ROMs/

ROM 1 : n.CPC/ROMs/Utopia.SysPatch.rom

ROM 2 : PC/ROMs/Maxam15.SysPatch.rom

ROM 3 : v.CPC/ROMs/Protext.SysPatch.rom

ROM 4 : ROMs/

ROM 5 : ROMs/

ROM 6 : Emulation:CPC/ROMs/Syspatch.rom

ROM 7 : Emulation:CPC/ROMs/Parados.rom

Désassembleur

MAME

Le debugger de MAME peut faire l'affaire

unununium

un-disas

Pas de debug symbolique

dasmxx

Désassembleur multi architectures

Utilisation d'un « listfile » permettant d'insérer des labels, commentaires, etc dans le code désassemblé

Petits problèmes pour le support de l'adressage 16 bit, patchs pas encore intégrés

Assembleur

XASM

Outil fourni avec u'nSP IDE

Pas de sources, Windows seulement

Syntaxe bizarre : R1 = R2 + [0x1234]

naken_asm

Assembleur multi machines

Permet uniquement de générer des fichiers binaires bruts (pas de format objet intermédiaire)

Vasm

Assembleur multi machine avec objets intermédiaires

Pas de support de l'adressage 16 bit et c'est compliqué à changer

Langages de haut niveau

Forth

Projet peu documenté et laissé à l'abandon

Java

Java Grinder : convertisseur de bytecode Java en assembleur

Performances incertaines

C

u'nSP IDE et son GCC pas libre

Vbcc, probablement le plus prometteur

Projet non libre :(

Pas d'utilisation commerciale, pas de diffusion des versions modifiées du compilateurs

Conclusion

État actuel

Portage de Contiki (OS temps réel minimaliste avec interface graphique)

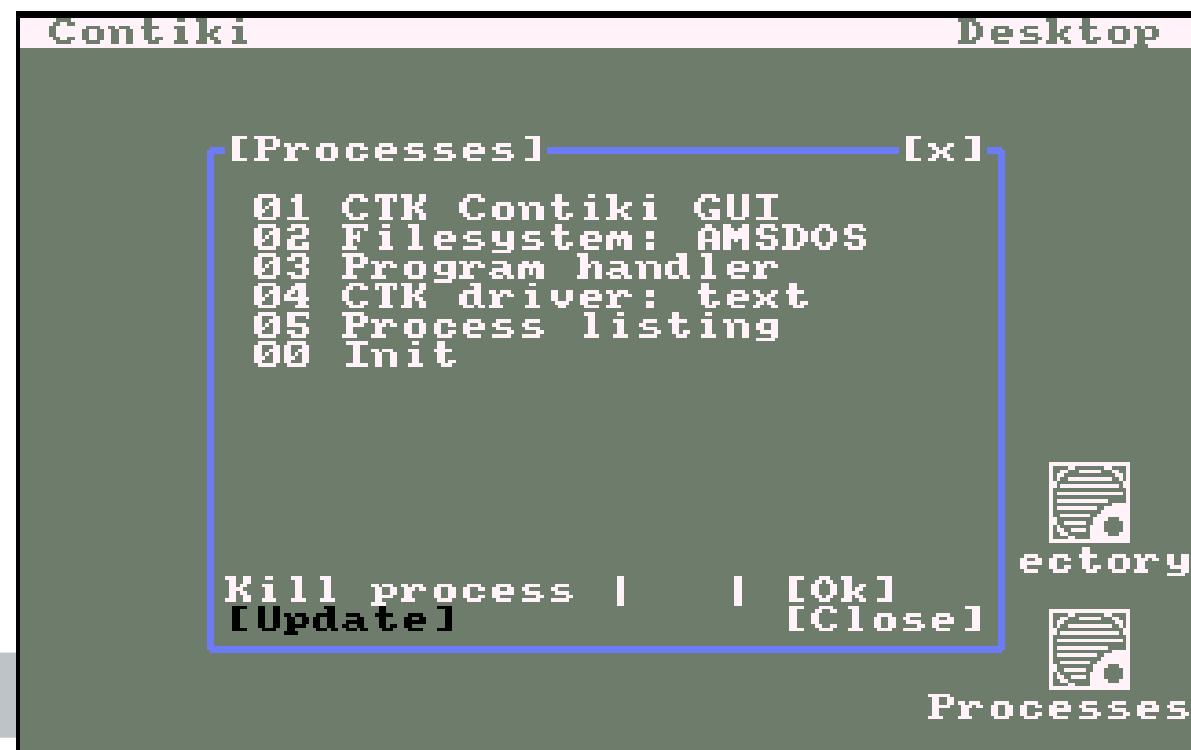
Le joystick ne fonctionne pas sur la vraie console (mais fonctionne dans MAME)

Quelques exemples simples

« Hello World »

Début d'animation de sprites

Tests pour les joysticks (pas fiable)



Prochaines étapes

Poursuivre le désassemblage du BIOS

Y-a-t'il un mode de test caché accessible par une combinaison de touches ?

Comprendre l'algo de checksum pour les contrôleurs

Faire un adaptateur pour utiliser les contrôleurs sur un PC

Documenter correctement la partie son

Écrire un lecteur de musique « module » MOD, XM, IT ou S3M

Faire marcher l'écran interne de la V.Smile Pocket

Seulement la sortie composite fonctionne pour l'instant

Continuer d'améliorer le compilateur C

Optimisations pour utiliser les modes d'adressage avec auto-incrémentation par exemple

Questions ?

Plus d'informations : <https://vtech.pulkomandy.tk>