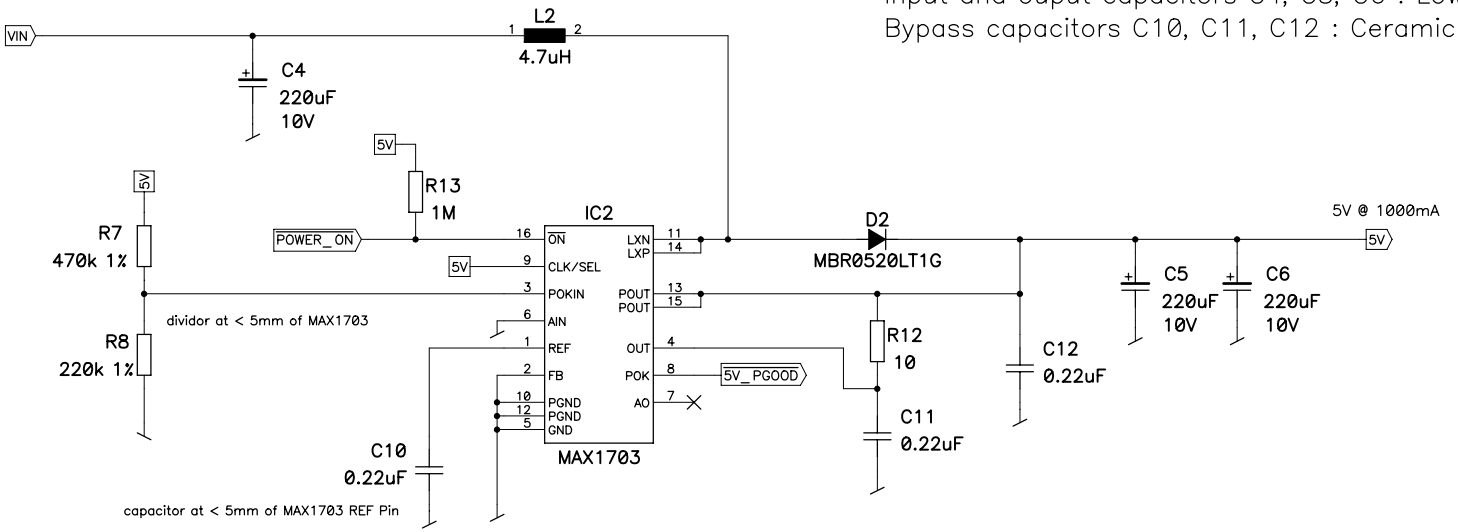
5V Power Supply

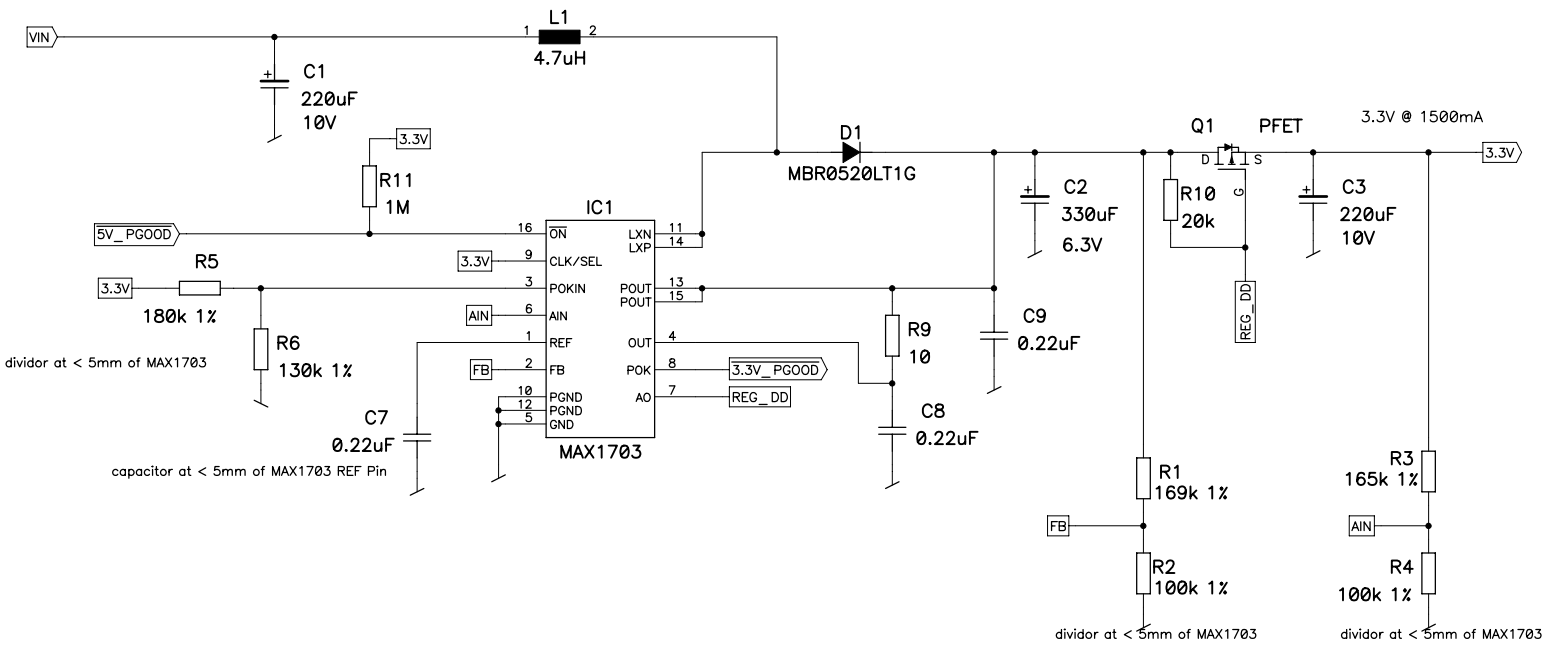
Input and ouput capacitors C4, C5, C6 : Low ESR Tantalum
Bypass capacitors C10, C11, C12 : Ceramic



iPresenter 2	DES	23.11.2007 Fournier
5V Power Supply	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	1/29	DATA02\ PowerSupply_TestBoard.sch

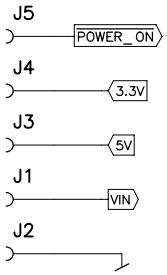
3.3V Power Supply

Input and ouput capacitors C1, C2, C3 : Low ESR Tantalum
Bypass capacitors C7, C8, C9 : Ceramic



iPresenter 2	DES	23.11.2007 Fournier
3.3V Power Supply	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	2/29	DATA02\ PowerSupply_TestBoard.sch

Test Connectors

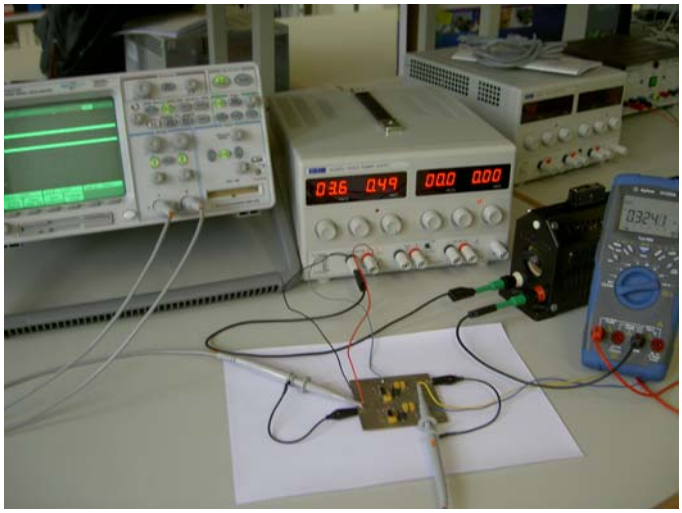


iPresenter 2	DES	23.11.2007 Fournier
Test Connectors	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	3/29	DATA02\ PowerSupply_TestBoard.sch

Procédure de test de l'alimentation réalisée

Protocole de test

Afin de valider le fonctionnement correct des alimentations, un petit protocole de test a été mis en place. Une alimentation linéaire de laboratoire simule l'accumulateur. Sa tension a été fixée à 3.6V, comme le serait l'accu durant la majorité de sa phase de décharge. L'alimentation est chargée via une résistance de puissance variable. L'oscilloscope mesure la tension à l'entrée et à la sortie. Le multimètre permet de mesurer le courant en entrée et en sortie de l'alimentation.



Lors des mesures, il faut bien veiller à placer la masse de la sonde proche du point de mesure. Autrement, d'autres bruits présents dans le plan de masse viendront s'ajouter aux mesures.

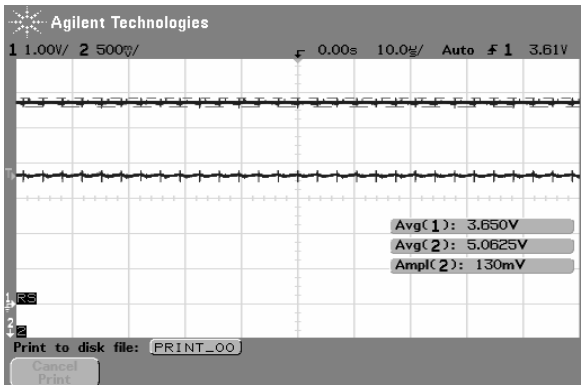
Résultat des mesures

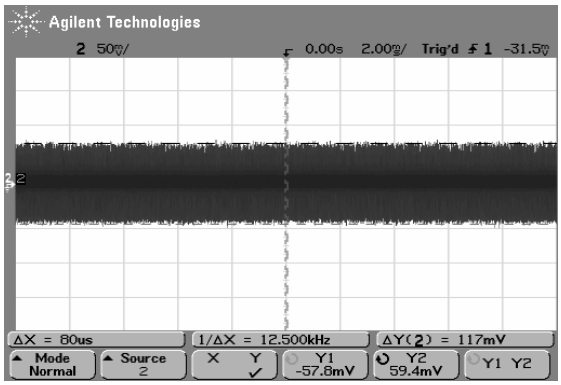
Voici un tableau récapitulatif des mesures effectuées. Les champs Pin, Pout et le rendement ont été calculés sur la base des autres champs.

Vin	Iin	Pin	Vout	Iout	Pout	Rendement	Bruit
3.6V	240mA	860mW	5.06V	150mA	760mW	88%	117mVpp
3.6V	380mA	1370mW	5.05V	250mA	1260mW	91%	137mVpp
3.6V	1580mA	1580mW	5.01V	1000mA	5000mW	88%	218mVpp

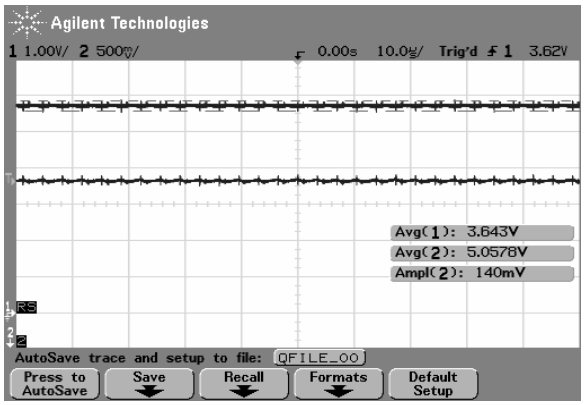
On peut constater que, même lorsqu'un courant de 1000mA est tiré sur l'alimentation 5V, le bruit généré à la sortie est inférieur aux tolérances (+/- 250mV). Le rendement est de l'ordre de 90%, ce qui est également bon.

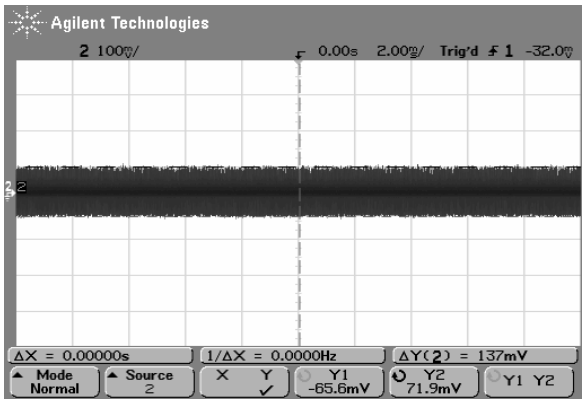
Iout = 150mA



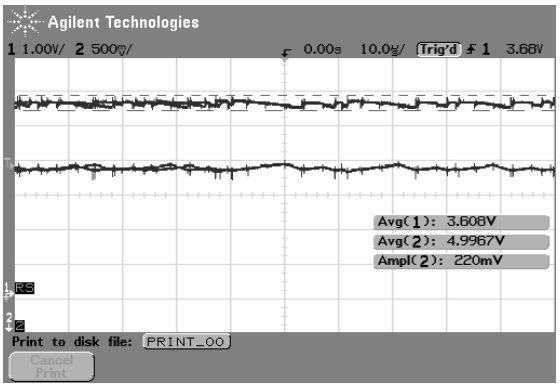


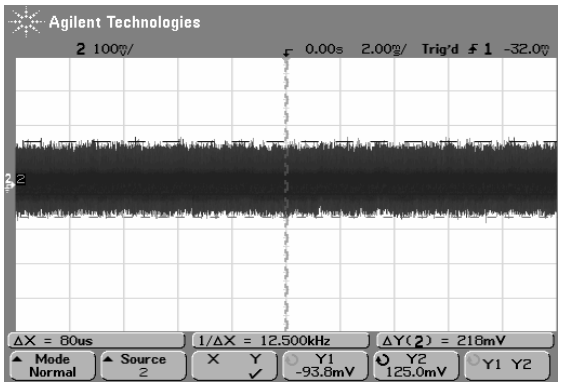
Iout = 250mA





Iout = 1000 mA

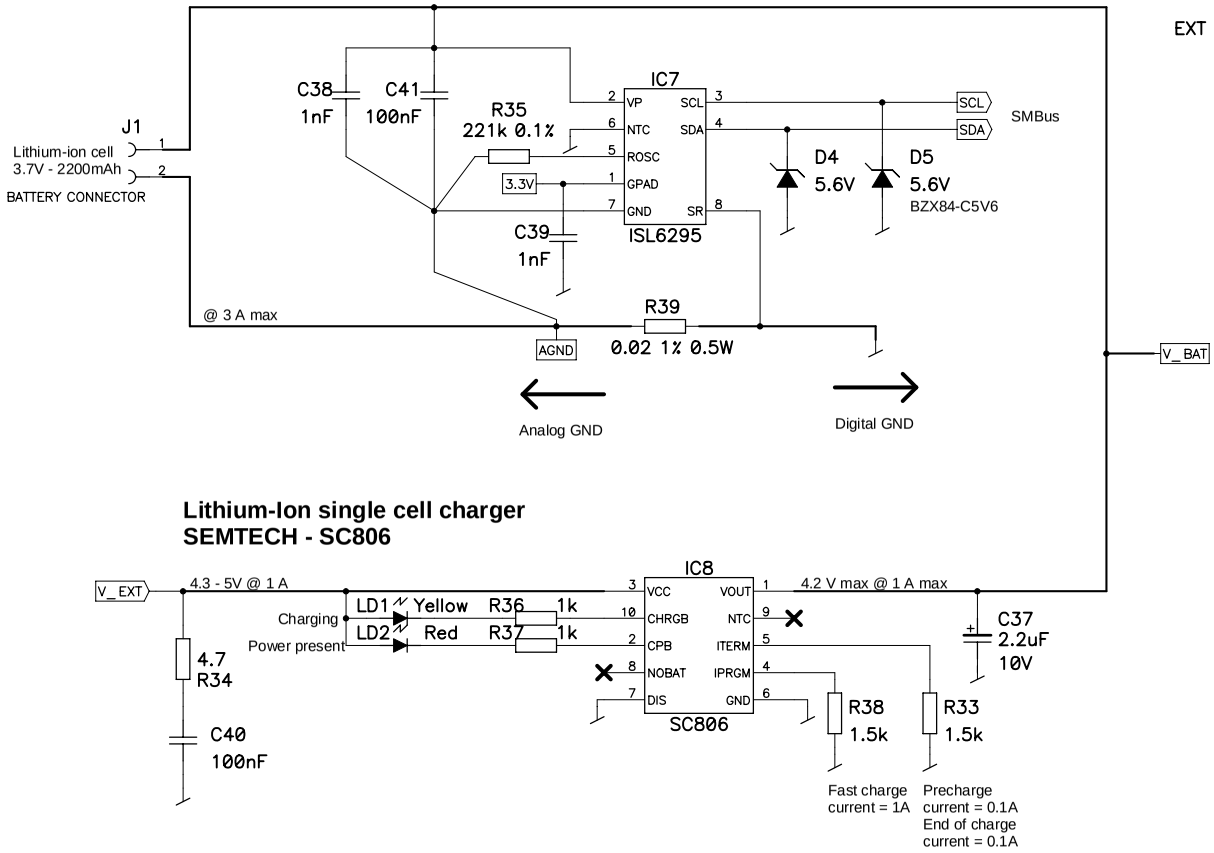


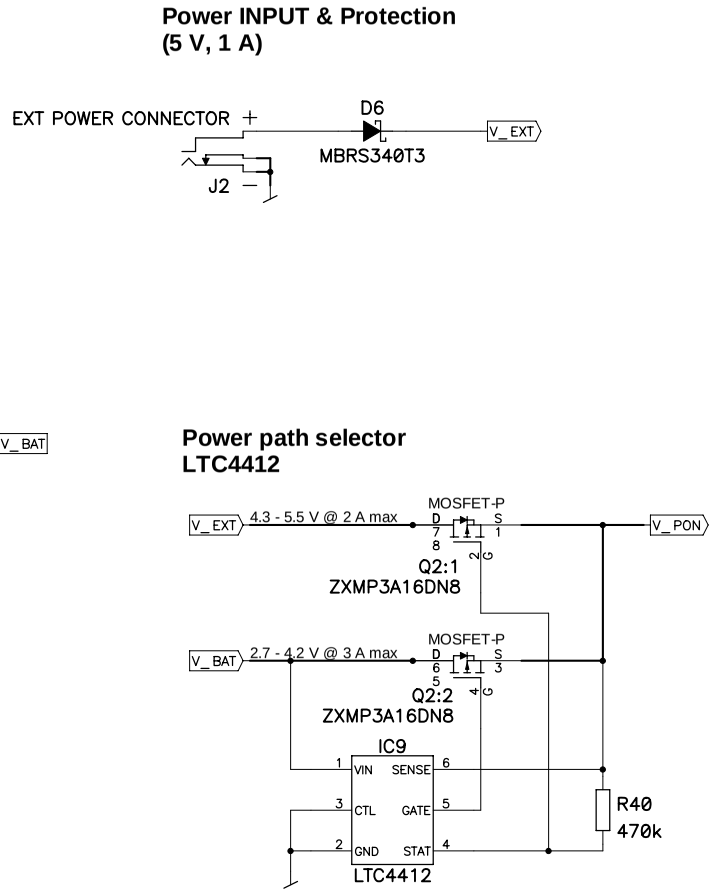


Pour ce qui est de l'étage 3.3V, les résultats sont aussi réjouissants. En effet, tant que la tension d'entrée est supérieure à 3.4V, le circuit de découpage est désactivé et seul le transistor limite la tension en sortie. Cela permet d'observer une tension tout à fait lisse et sans bruit à la sortie. L'échauffement de celui-ci est très faible, même avec une tension d'entrée de 4.2V. La puissance maximale dissipée est de l'ordre de : P = U * I = (4.2 – 3.3) * 0.38A = 342mW au maximum. (le transistor supporte 500mW).

Lorsque la tension descend en dessous de 3.4V, le découpage commence et le bruit observable est similaire à celui de la ligne 5V et toujours dans les tolérances. Le rendement observé moyen est d'environ 80% pour cette ligne.

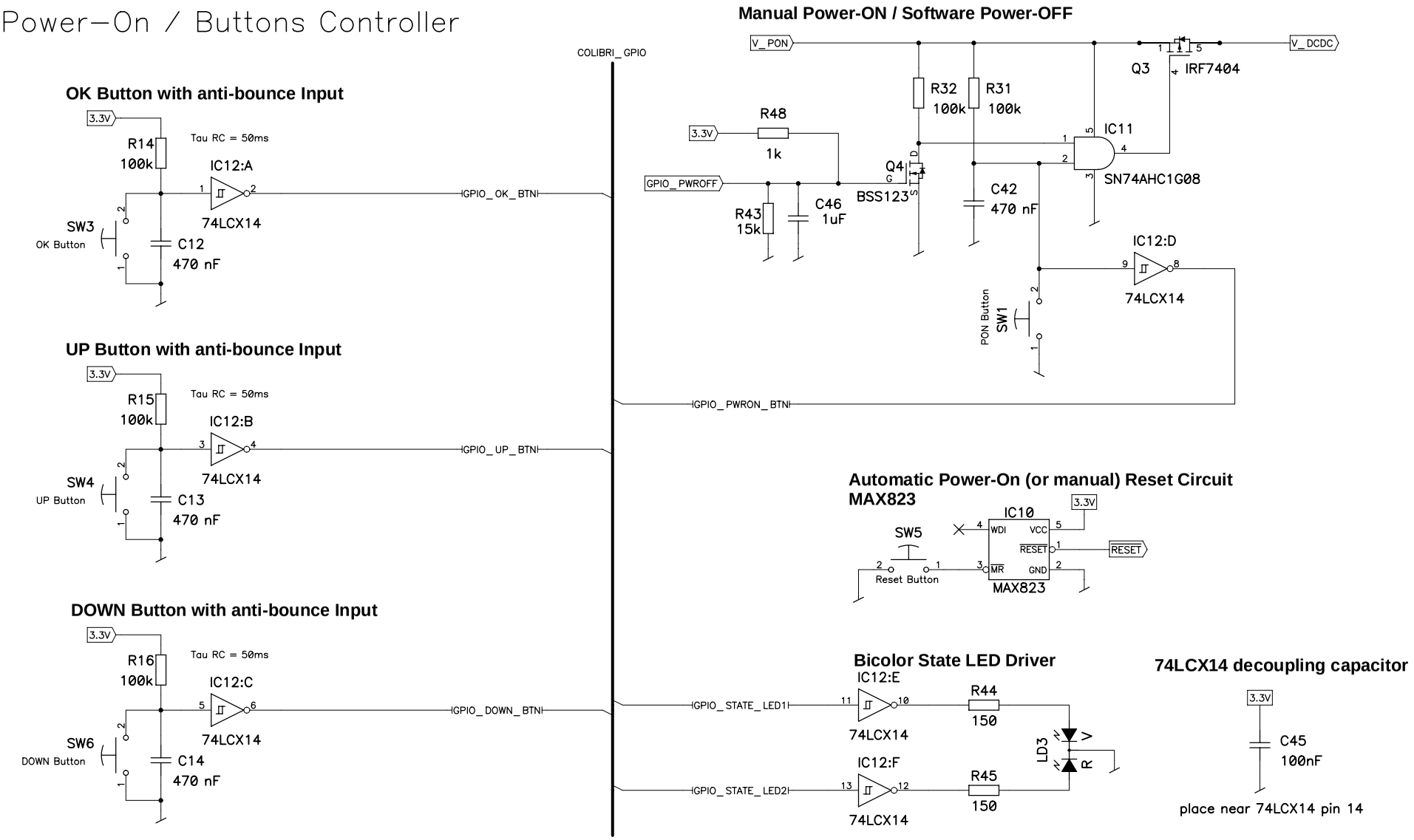
Battery Management – Charger / Measure





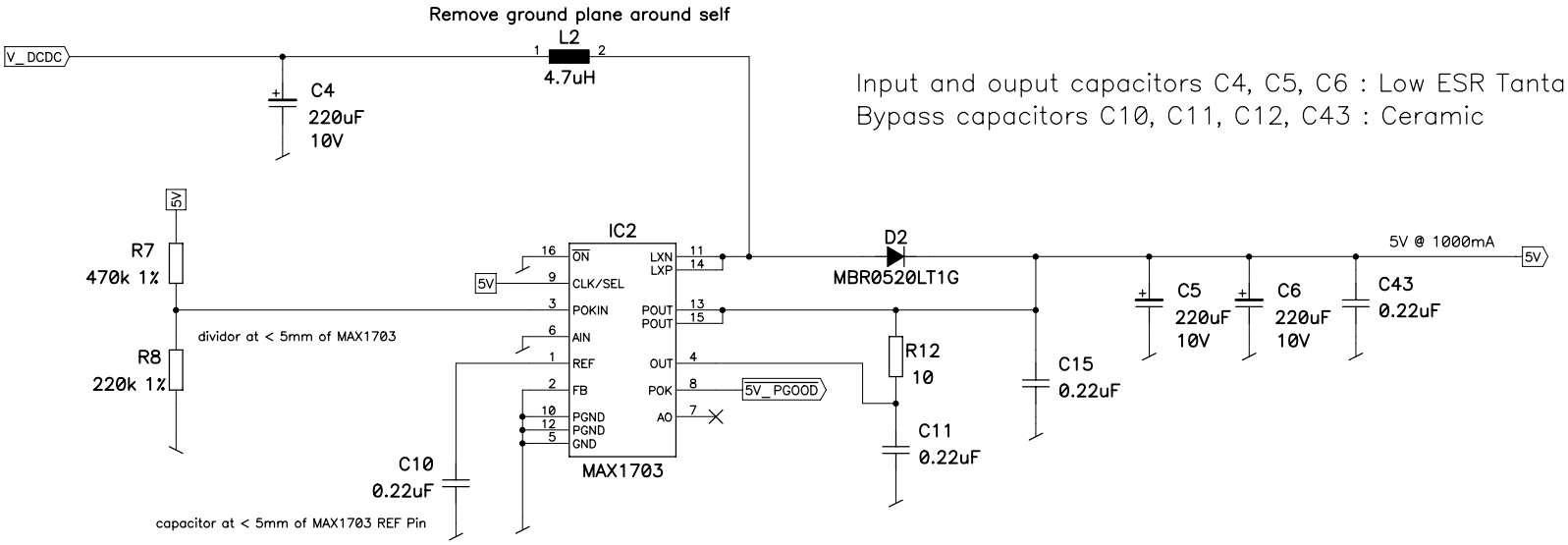
iPresenter 2	DES	23.11.2007 Fournier
Battery Management	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	1/8	DATA02\ iPresenter2.sch

Power-On / Buttons Controller



iPresenter 2	DES	23.11.2007 Fournier
Power-On / IOs	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	2/8	DATA02\ iPresenter2.sch

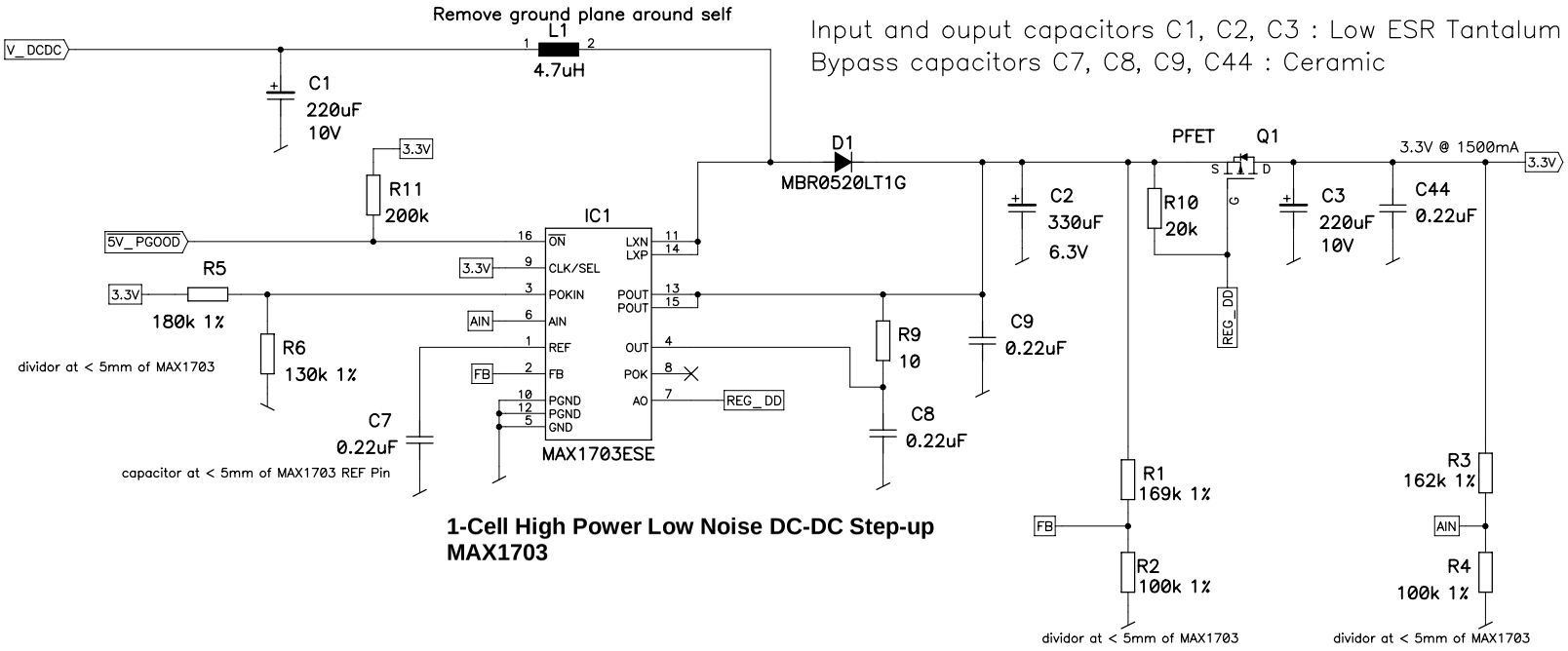
5V Power Supply



1-Cell High Power Low Noise DC-DC Step-up MAX1703

iPresenter 2	DES	23.11.2007 Fournier
5V Power Supply	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	3/8	DATA02\ iPresenter2.sch

3.3V Power Supply

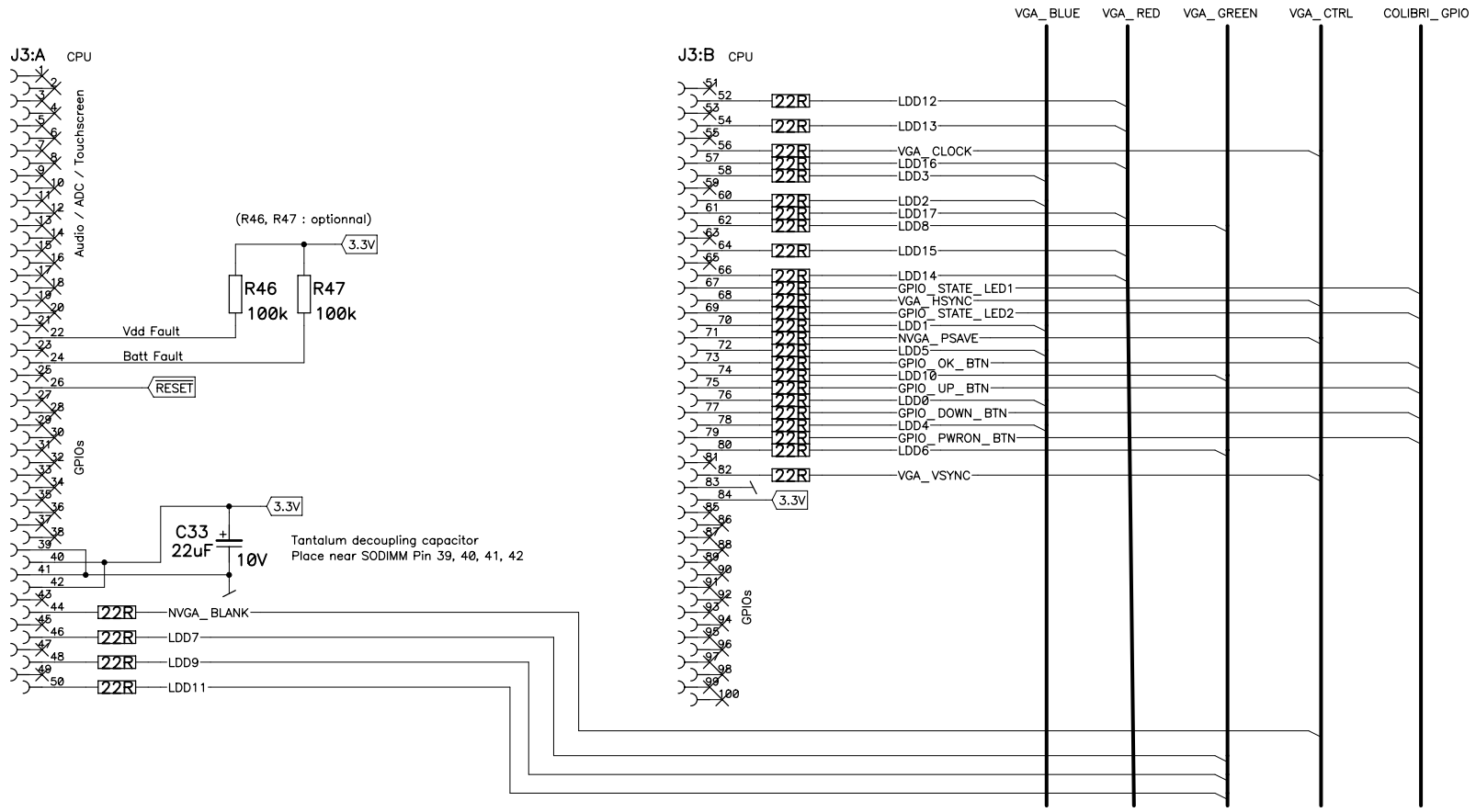


Input and ouput capacitors C1, C2, C3 : Low ESR Tantalum
Bypass capacitors C7, C8, C9, C44 : Ceramic

1-Cell High Power Low Noise DC-DC Step-up MAX1703

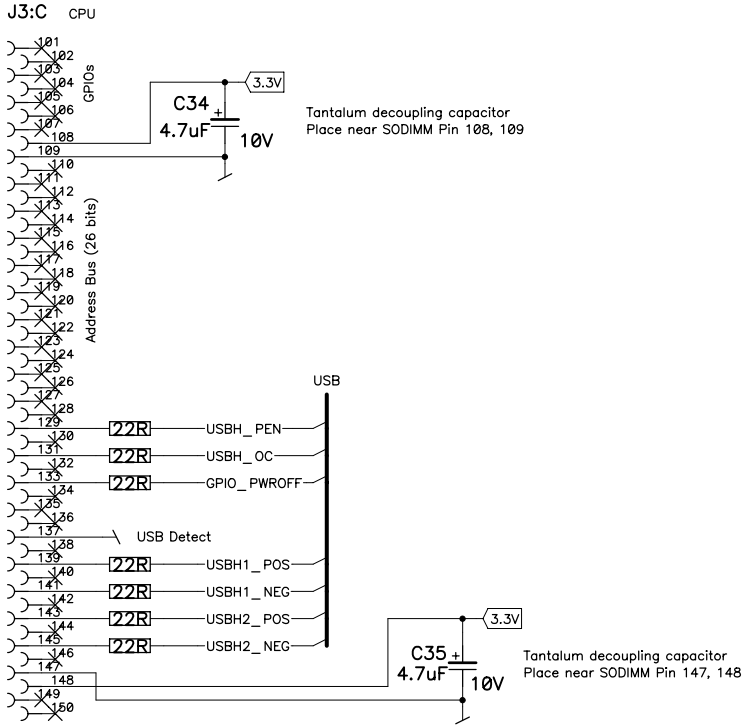
iPresenter 2	DES	23.11.2007 Fournier
3.3V Power Supply	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	4/8	DATA02\ iPresenter2.sch

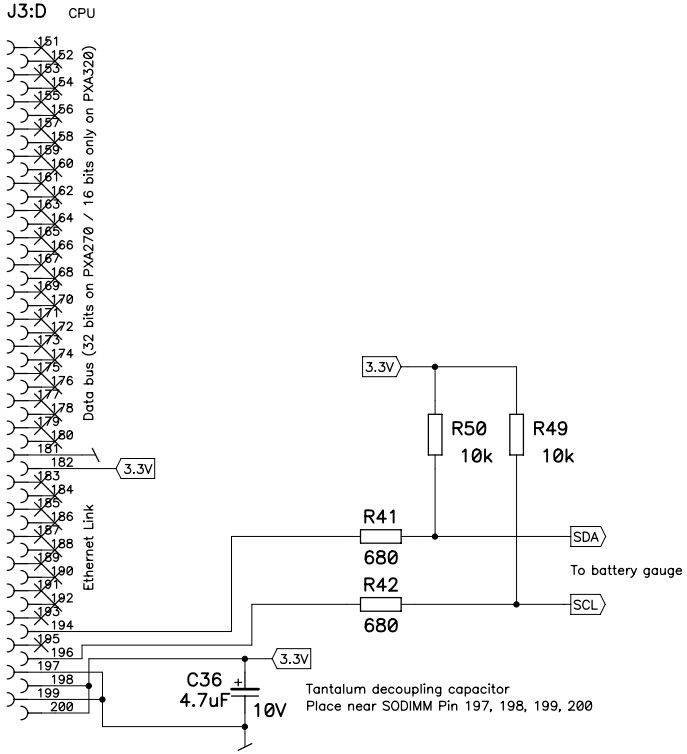
Colibri Connector (1 / 2)



iPresenter 2	DES	23.11.2007 Fournier
CPU 1/2	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	5/8	DATA02\ iPresenter2.sch

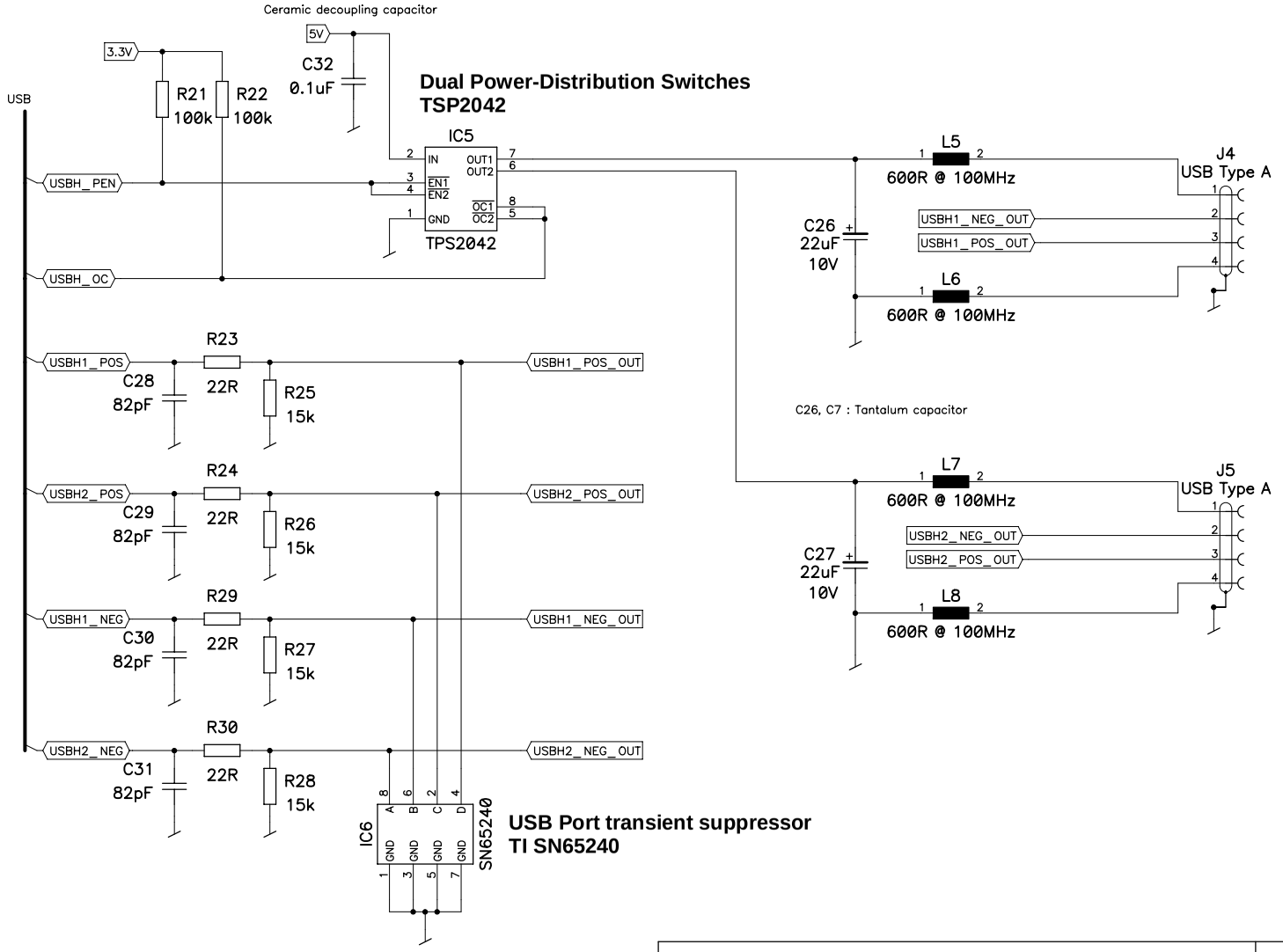
Colibri Connector (2 / 2)





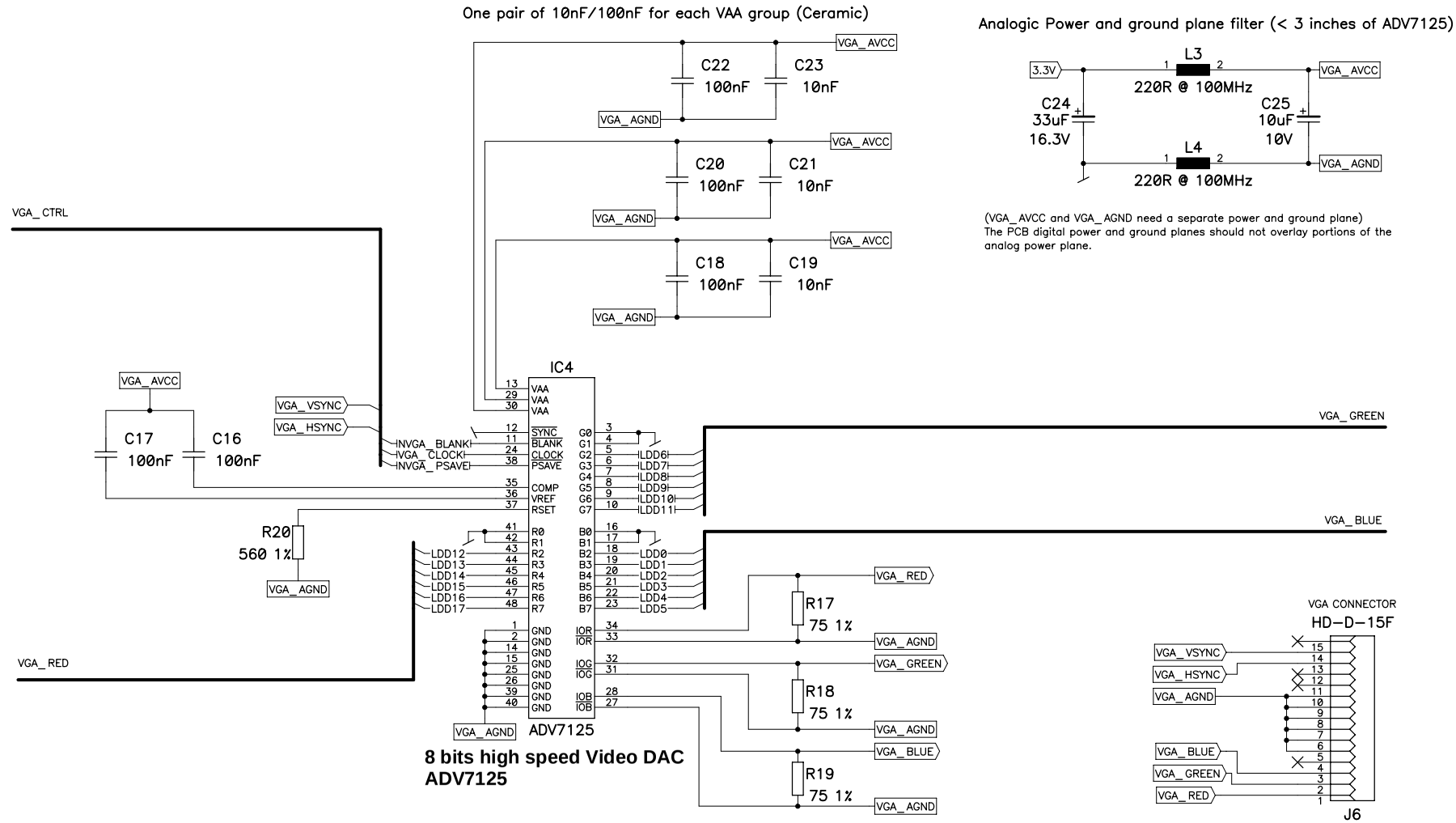
iPresenter 2	DES	23.11.2007 Fournier
CPU 2/2	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	6/8	DATA02\ iPresenter2.sch

USB Output



iPresenter 2	DES	23.11.2007 Fournier
USB Output	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	7/8	DATA02\ iPresenter2.sch

Video DAC and output



iPresenter 2	DES	23.11.2007 Fournier
Video DAC	REV	REVISION 1
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	8/8	DATA02\ iPresenter2.sch

iPresenter2 v1.0 - Listing Composants

Reçu commande	
En stock HEVs	

Battery Charger

Référence	Description	Valeur	Constructeur	Référence	Fourniseur	Ref. Fourn.	Quant.	Prix carte
J1	Connecteur Molex vertical 2 voies	-	MOLEX	22035025	Farnell	9979611	2	0.375
BATT	Batterie Li-Ion LIP-18650 3.7V	2200mAh	VARTA	56620201098	Farnell	5088252	1	53.35
C37	Condensateur Tantale Case A 10V	2.2uF	EPCOS	B45196E2225K109	Farnell	9753796	2	0.627
C38, C39, C40, C41	Condensateur céramique Case 805	**	-	-	HEVs	-		0
R33, R36-R38, R40	Résistance SMD 603	**	-	-	HEVs	-		0
R34	Résistance SMD 805	4.7R	-	-	HEVs	-		0
LD1	LED 3mm jaune	-	-	-	HEVs	-		0
LD2	LED 3mm rouge	-	-	-	HEVs	-		0
R35	Résistance SMD 805 0.1%	221k	WELWYN	PCF0805R 221KBI.T1	Farnell	1160277	2	1.45
R39	Résistance Metal Strip 2010 1% 0.5W	0.02R	VISHAY	WSL-2010 .02 1%	Farnell	1107366	2	2.02
D4, D5	Diode Zener 350mW SOT-23	5.6V	FAIRCHILD	BZX84-C5V6	Farnell	9844406	4	0.53
Q2	Transistor MOS-FET double canal SO-8	-	FAIRCHILD	FDS6875	Farnell	9844899	2	2.95
J2	Connecteur DC 2.5MM PQ10	-	CLIFF	DC10B	Farnell	224960	10	1.82
IC7	Gauge batterie Lithium Ion ISL6295	-	INTERSIL		Digikey	--	1	2.75
IC8	Chargeur batterie Lithium Ion SC806	-	SEMTECH		Digikey	--	1	7.85
IC9	Comm. automatique de source LTC4412	-	LINEAR		Digikey	--	1	2.85
D6	Diode Shottky 3A 40V	-	ON	MBRS340T3G	Farnell	9557377	2	1.2

Alimentation

Référence	Description	Valeur	Constructeur	Référence	Fourniseur	Ref. Fourn.	Quant.	Prix carte
L1, L2	Inductance Coiltronics DR-73	4.7uH	COILTRONICS	DR73-4R7-R	Farnell	9796355	4	4.2
D1, D2	Diode Schottky 0.5A 20V	-	ON SEMICOND.	MBR0520LT1G	Farnell	9556915	4	1.12
IC1, IC2	Convertisseur DC/DC Maxim 1703	-	MAXIM DALLAS	MAX1703ESE+	Farnell	9725075	3	28.5
Q1	MOSFET Canal-P	-	FAIRCHILD	FDN338P	Farnell	9846301	2	2.1
C1, C3, C4-C6	Cond. low ESR Tantalum Case D 10V	220uF	AVX	TPSD227K010R0100	Farnell	1135110	15	12
C2	Cond. low ESR Tantalum Case D 6.3V	330uF	AVX	TPSD337K006R0045	Farnell	1135198	4	3.05
C7-12, C43	Cond. céramique 25V	0.22uF	EPCOS	B37931K0224K60	Farnell	8805644	30	0.644
R1 - R9, R12	Résistance SMD 603 1%	**	-	-	HEVs	-	20	0
R10, R11	Résistance SMD 603	**	-	-	HEVs	-	6	0

CPU

Référence	Description	Valeur	Constructeur	Référence	Fourniseur	Ref. Fourn.	Quant.	Prix carte
J_CPU	Support CI DIMM 200 Pins DDR1 2.5V	-	TYCO	1473005-1	Farnell	1095876	2	11.25
C33	Condensateur Tantale Case B 10V	22uF	AVX	TPSB226K010R0700	Farnell	1135106	2	0.65
C34, C35, C36	Condensateur Tantale Case B 10V	4.7uF	EPCOS	B45196E2475K209	Farnell	9753800	6	2.7
RES1-RES9	Réseau 4 résistances 603	22R	PHYCOMP	235002512209	Farnell	1086521	20	5.3
R41, R42, R46, R47	Résistance SMD 603	**	-	-	HEVs	-		0

Button, Power On & Reset

Référence	Description	Valeur	Constructeur	Référence	Fourniseur	Ref. Fourn.	Quant.	Prix carte
SW1-SW6	Interrupteur impulsionel	-	-	-	HEVs	-	10	0
IC12	Six inverseurs à trigger de schmitt	-	Fairchild	74LCX14M	Farnell	1013778	2	1.1
R14-R16, R31, R32, R48	Résistances SMD 603	**	-	-	HEVs	-	10	0
R44, R45	Résistances SMD 603	150	-	-	HEVs	-	4	0
C12-C14, C42	Condensateur céramique Case 603	500nF	-	-	HEVs	-	6	0
IC10	Superviseur micro-processeur	-	MAXIM	MAX823SEUK+T	Farnell	1187963	2	4.75
IC11	Porte AND 2 entrées	-	TI	SN74AHC1G08DCKRG4	Farnell	1287466	2	0.398
Q3	Transistor MOSFET canal P 5.3A	-	IR	IRF7404PBF	Farnell	9103384	2	5.55
Q4	Transistor MOSFET canal N 0.17A	-	Fairchild	BSS123	Farnell	9845321	2	0.51
LD3	LED 3mm bicolore	-	Kingbright	L-93WEGW	Farnell	1142520	2	0.928
C45, C46	Condensateur céramique Case 805	**	-	-	HEVs	-		0

Video

Référence	Description	Valeur	Constructeur	Référence	Fourniseur	Ref. Fourn.	Quant.	Prix carte
IC4	Convertisseur Video D/A ADV7125	-	ANALOG DEV.	ADV7125KSTZ140	Farnell	1319340	2	24.1
J_VGA	Connecteur HDD-SUB 15 p. VGA Travers.	-	TYCO / AMP	1734530-3	Farnell	1056018	2	16.9
C24	Condensateur Tantale Case B 16.3V	33uF	AVX	TAJB336K016R	Farnell	1135054	2	0.84
C25	Condensateur Tantale Case A 10V	10uF	AVX	TCJA106M010R0300	Farnell	1135234	2	0.84
L3, L4	Perle de ferrite 500mA Case 603	600R @ 100M	TDK	MMZ1608Y601BTA00	Farnell	1301668	4	0.804
R17-20	Résistances SMD 603 1%	**	-	-	HEVs	-	5	0
C16-18, C20, C22	Condensateur céramique Case 603	100nF	-	-	HEVs	-	10	0
C19, C21, C23	Condensateur céramique Case 603	10nF	-	-	HEVs	-	6	0

USB

Référence	Description	Valeur	Constructeur	Référence	Fourniseur	Ref. Fourn.	Quant.	Prix carte
IC5	Commutateur distrib. Double TI 2042	-	TI	TPS2042BD	Farnell	8456755	4	2.05
IC6	Suppresseur pour port USB double	-	TI	SN65240PW	Farnell	8452016	2	0.83
J_USB1, J_USB2	Connecteur USB Type A Travers.	-	Lumberg	2410 02	Farnell	1177883	4	1.98
L5, L6, L7, L8	Perle de ferrite 500mA Case 603	600R @ 100M	TDK	MMZ1608Y601BTA00	Farnell	1301668	8	1.608
C26, C27	Condensateur Tantale Case A 10V	22uF	AVX	TAJA226K010R	Farnell	1135050	4	1.68
R21, R22, R49, R50	Résistances SMD 603	**	-	-	HEVs	-	4	0
R23-R30	Résistances SMD 603 1%	**	-	-	HEVs	-	14	0
C28-31	Condensateur céramique Case 603	82pF	-	-	HEVs	-	8	0
C32	Condensateur céramique Case 603	100nF	-	-	HEVs	-	2	0

TOTAL	**214.154**

iPresenter2 v1.0 - Procédure de test

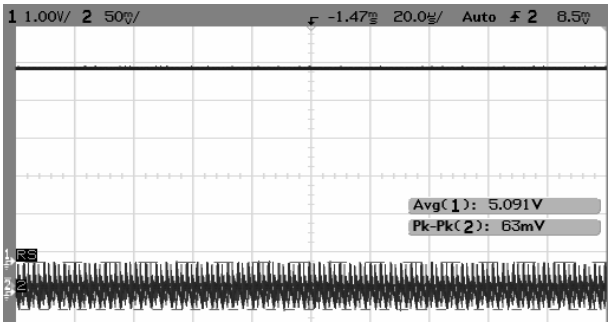
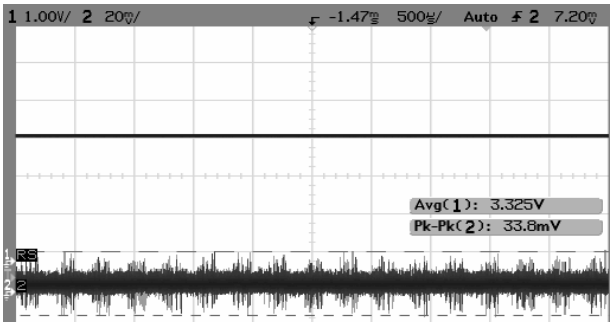
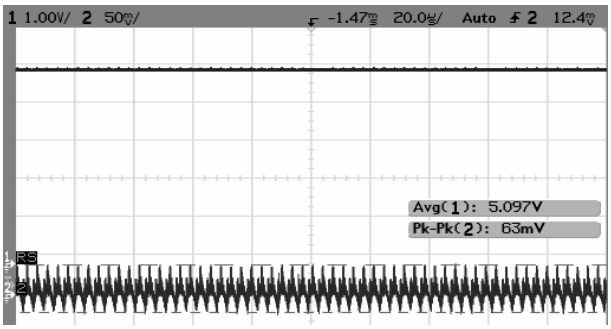
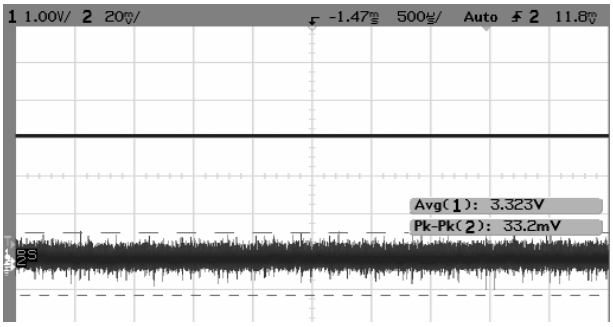
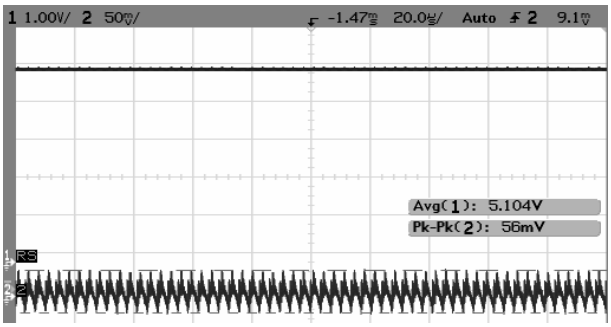
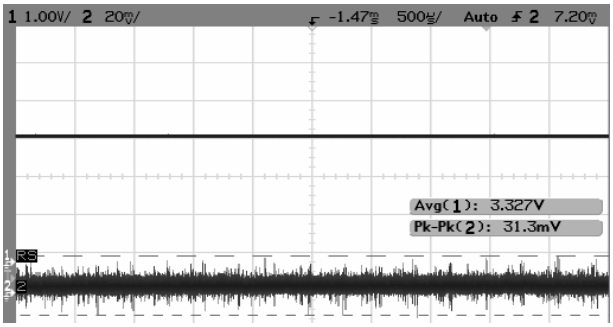
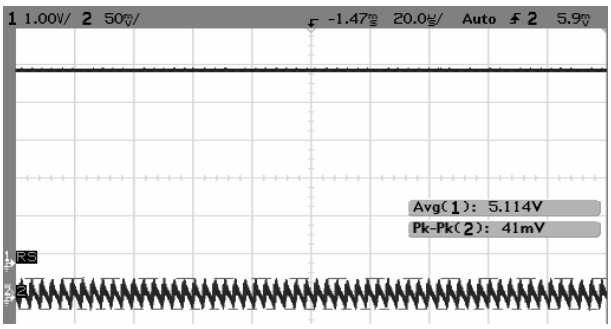
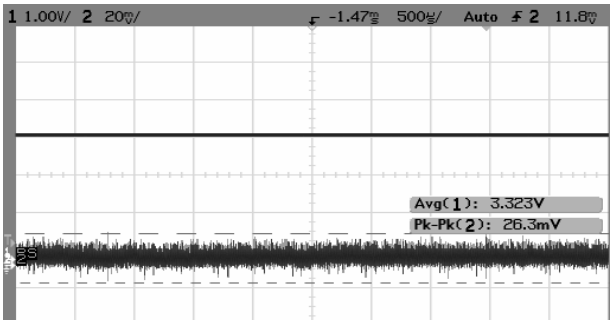
Cat.	N°	Description	Résultats attendus	Résultats analysés	Mesures prises
Chargeur Batterie		TOUS LES COMPOSANTS SOUDES, BATTERIE DEBRANCHEE, CARTE COLIBRI RETIREE, TOUS CONNECTEURS LIBRES			
	1	Test de court-circuit entre les bornes + et - de la batterie (ohmmètre)	Pas de court-circuit	52 Mohms	-
	2	Le côté négatif du connecteur batterie est relié au plan de masse de la carte	0.02 ohms à l'ohmmètre	0.025 Ohms	-
		CONNEXION DE LA BATTERIE / ALIMENTATION DE LABORATOIRE 3.6V AVEC LIMITATION DE COURANT (afin d'éviter un court-circuit desctructeur)			
	3	Il ne doit pas y avoir de tension après Q3	0V sur les pins 4,5,6,7 de Q3	0.02V	-
	4	La tension de la batterie se retrouve sur les pins 5 et 6 de Q2	3.6V sur les pins 5-6 de Q2	3.64V	-
	5	La tension de la batterie se retrouve sur les pins 1,2,3 de Q3	3.6V sur la pin 1,2,3 de Q3	3.64V	-
	6	Test préliminaire de IC7 : Alimentation et masse	3.6V sur pin 2 et 0V sur pin 7 et 8	3.65V et 0.02V	-
	7	Test prélimintaire de IC8 : Alimentation et masse	3.6V sur pin 1 et 0V sur pin 7 et 8	3.65V et 0.02V	-
		CONNEXION DU CHARGEUR Vext = 4.5V Ilimit = 1A (remplacer l'alimentation de laboratoire par la batterie avant, obligatoire)			
	8	Test visuel : LD1 et LD2 doivent s'allumer pour indiquer la charge	LD1 et LCD2 sont allumés	LD1 et LD2 brillent	-
	9	Mesure de la tension sur R39 pour vérifier que la batt. se charge	Tension R39 non nulle et max. 20mV	7mV	-
	10	Attendre la fin de la charge pour vérifier l'arrêt automatique	Tension R29 = 0V et LD1 éteinte	LD1 s'éteint après 1h	-
	11	Lors de la connexion du chargeur, pin 1,2 et 3 de Q3 à Vext (P. Path)	pin 1, 2 et 3 de Q3 = 4.8V	4.78V	-
		DECONNEXION DU CHARGEUR, RECONNEXION DE L'ALIMENTATION DE LABORATOIRE (au besoin)			
Power-On	12	Présence de la tension batterie sur IC11 et la base de Q4	3.6V sur IC11 pin 1,2,5 et Q4:D	3.62V	-
	13	La porte AND IC11 doit avoir en sortie la valeur 1	3.6V sur IC11 pin 4	3.64V	-
	14	Lors de l'appui sur le bouton SW1, Q3 doit se mettre à conduire	3.6V sur Q3 pin 4,5,6,7,8	3.59V	-
	15	Lors du relâchement du bouton SW1, Q3 ne conduit plus	0V sur les pins 4,5,6,7 de Q3	0.04V	-
Power supplies		MAINTENIR SW1 ENLENCHE			
	16	Présence de la tension batterie à l'entrée des 2 convertisseurs DC/DC	3.6V sur la pin 1 de L1 et de L2	3.59V	-
	17	Vérifier les tensions à la sortie des convertisseurs	3.3V sur C3+ et 5V sur C5+		-
	18	Vérifier les tolérances et bruits à la sortie des convertisseurs	+/- 5% sur le 5V et +/- 5% sur le 3.3V	Chap. 9.1.2 / page xx	-
	19	Vérifier qu'en faisant varier la tension d'entrée, les sorties restent stables	Varier Vbatt de 2.8V à 4.2V, contrôler sorties		-
	20	Vérifier les tolérances et bruits lors de variations de tension d'entrée	+/- 5% sur le 5V et +/- 5% sur le 3.3V		-
Buttons & PowerU	21	Tensions d'entrée du trigger IC12 (alim. 3.3V / entrée bouton UP, DOWN, OK)	IC12 pin 1,3,5,14 à 3.3V	3.56V et 0.03V	-
	22	Tension d'entrée du trigger IC12 (bouton SW1)	IC12 pin 9 à Vbatt	3.54V	-
	23	Appui sur les boutons UP, DOWN et OK, sorties sans rebond de IC12 (oscillo.)	IC12 pins 2,4,6,8 flancs nets sans rebonds	Chap. 9.1.3	-
	24	Forcer les entrées de IC12 pour vérifier l'allumage de la led bicolore	Quand IC12 pin 11 et 13 à 3.3V => LEDs ON	les LEDs s'allument	-
	25	Vérifier lors du démarrage du 3.3V que le reset aie bien lieu sur IC10	Après démarrage 3.3V, impuls. Sur IC10 pin 1	Chap. 9.1.4	-
	26	Vérifier que le bouton reset manuel fonctionne	Appui sur SW5, impuls. Sur IC10 pin 1	Le reset fonctionne	-
	27	Vérifier le maintient de tension par le processeur, en forcant 3.3V sur R43 pin 1	Coupure de l'alimentation 3.3V et 5V	L'alim. Est bien maintenue	-

USB	28	La tension de 5V est bien présente sur l'IC de distribution IC5	IC5 pin 2 à 5V	5.05V	-
	29	Les deux signaux d'activation d'IC5 sont bien à l'état haut (désactivé)	IC5 pin 3 et 4 à 3.3V	3.52V	-
	30	Forcer la pin 3 d'IC5 à 0V et vérifier la présence du 5V sur le connecteur	Connecteurs USB J4 et J5 Pin 1 à 5V	5.04V sur les conn.	-
	31	Tirer plus que 0.5A de courant sur les connecteurs et vérifier le signal OC1 et OC2	OC1 et OC2 passent à 0 lorsque I > 0.5A	0.02V pour I > 0.56A	-
Display	32	Contrôler l'alimentation analogique du convertisseur vidéo IC4	3.3V entre les pins de C24	3.53V	-
	33	Contrôler les groupes d'alimentation du convertisseur vidéo IC4	3.3V sur les pins 13, 29 et 30 d'IC4	3.51V / 3.51V / 3.53V	-
	34	Vérifier les masses analogiques du convertisseur vidéo IC4	0V sur les pins 1,2,14,15,25,26,39,40, 27,31,33	tout à ~0.04V	-
	35	Vérifier les masses de signaux du convertisseur vidéo IC4	0V sur les pins 12, 3,4,15,16,41,42	tout à ~0.02V	-
	36	Vérifier la connexion correcte des signaux entre IC4 et le connecteur VGA	IC4:34 > J6:1 / IC4:32 > J6:2 / IC4:28 > J6:3	Problèmes de connexions	**soudures**
	37	Vérifier la connexion à la masse de la résistance Rset	R20 pin 2 à 0V	0.02V sur Rset pin 2	-
	38	Vérifier la connexion au 3.3V analogique des condensateurs de réglage d'IC4	C16 et C17 pin 1 à 3.3V	3.51V	-
CPU Connector	39	Vérifier les masses du connecteur CPU	J3 pin 39,41,83,109,147,181,197,199 à 0V	0.02V	-
	40	Vérifier les alimentations du connecteur CPU	J3 pin 40,42,84,108,148,182,198,200 à 3.3V	3.54V	-
	41	Vérifier que les signaux d'alarme Batterie et défaut soient désactivés	J3 pin 22, 24 à 3.3V	3.54V	-
	42	Contrôler la connexion des signaux USB	J3 pin 129, 131, 139 à 145	OK	-
	43	Contrôler la connexion des signaux SMBUS	J3:194 > IC7:4 / J4:196 > IC7:3	OK	-
	44	Vérifier la connexion des signaux des boutons et LED au trigger IC12	J3:67,69,73,75,77,79,137 à IC12	OK	-
	45	Vérifier la connexion des signaux graphiques à IC4 et J6	J3:44,46,48,50,52,54,56,58,60,61,62,64,66,…	Problèmes de connexions	**soudures**
	46	Vérifier que les autres pins du connecteur J3 soient libres de potentiel	J3:1-21,27-38, 85-100, 101-108, 110-128, …	OK	-
FCT. / ALIM LOAD		CONNECTER LE MODULE COLIBRI SI TOUS LES TESTS ONT REUSSI (WINCE 5 démarre)			
		Vérifier les tensions à la sortie des convertisseurs	3.3V sur C3+ et 5V sur C5+	Chap. 9.1.2	-
		Vérifier les tolérances et bruits à la sortie des convertisseurs	+/- 5% sur le 5V et +/- 5% sur le 3.3V		-
		Vérifier qu'en faisant varier la tension d'entrée, les sorties restent stables	Varier Vbatt de 2.8V à 4.2V, contrôler sorties		-
		Vérifier les tolérances et bruits lors de variations de tension d'entrée	+/- 5% sur le 5V et +/- 5% sur le 3.3V		-
	47	Connecter un moniteur XVGA sur J6 et vérifier si l'image affichée est bonne	affichage 640x480 / bonnes couleurs / stable	OK	-
	48	Connecter un stick usb sur les 2 connecteurs USB et vérifier si elle est reconnue	Détection d'une clé par WIN CE 5.0 avec mess.	Chap. 9.1.5	
		Transférer des fichiers depuis la clé, connecter et tester souris / clavier	transfert de gros fichiers, souris, clavier reconnu	OK	-
		TESTS FONCTIONNELS TERMINES (BOUTONS ET PORT SMBUS A TESTER APRES DEVELOPPEMENT DE PILOTES)			

Procédure de test de l'alimentation finale de l'iPresenter2

Le premier test est effectué sans que le processeur ne soit connecté, comme défini dans la procédure de test (N° 17 à 20). Voici les résultats :

Mesure du 5V : aux bornes d'entrées 5V du distributeur double des ports USB (masse du port USB)
Mesure du 3.3V : aux bornes d'un des condensateurs de découplage du connecteur processeur (masse du connecteur)

Vbatt	Ligne 5V	Ligne 3.3V
2.8V		
3.3V		
3.6V		
4.2V		

Voici les valeurs des différentes courbes synthétisées dans un tableau :

Vbatt	Ligne 5V		Ligne 3.3V	
	Tension	**Bruit**	**Tension**	**Bruit**
2.8V	5.09V	63mVpp	3.32V	33.8mVpp
3.3V	5.10V	63mVpp	3.32V	33.2mVpp
3.6V	5.10V	56mVpp	3.33V	31.3mVpp
4.2V	5.11V	41mVpp	3.33V	26.3mVpp

La première constatation est que les tensions à vide sont très stables. A aucun moment le bruit sur la ligne 3.3V ne dépasse 35mVpp alors que la tolérance pour cette dernière est de 200mVpp (PXA320). De même pour la ligne 5V qui ne dépasse pas les 65mVpp de bruit alors que la limite est de 500mVpp.

Il faut aussi remarquer que le bruit à vide est plus de 2x moins élevé que sur la carte d'essai. Cela est dû à un bien meilleur routage du circuit et à la mesure effectuée aux bornes du distributeur double qui est lui aussi découplé à la masse par un condensateur.

Maintenant que nous savons que l'alimentation est capable de fournir les bonnes tensions de manière stable, il est possible de brancher les cartes Colibri, de les démarrer, et de ré-effectuer les mêmes tests. Ceux-ci sont d'autant plus importants qu'ils vont permettre d'assurer le bon fonctionnement de l'appareil dans tous les cas.

Mesure du 5V : sur les bornes d'un des connecteurs USB (la ligne USB étant chargée avec une souris optique et une clé mémoire USB)

Mesure du 3.3 V : sur une des pins d'alimentation du connecteur Colibri (avec cartes Colibri en place et démarrées)

Voici les résultats, cette fois-ci uniquement les valeurs chiffrées :

	Vbatt	Ligne 5V		Ligne 3.3V	
		Tension	**Bruit**	**Tension**	**Bruit**
PXA270	**2.8V**	5.08V	131mVpp	3.28V	92mVpp
	3.3V	5.08V	124mVpp	3.32V	81mVpp
	3.6V	5.08V	118mVpp	3.29V	55mVpp
	4.2V	5.08V	95mVpp	3.32V	53mVpp
PXA320	**2.8V**	5.04V	132mV	3.24V	137mV
	3.3V	5.03V	123mV	3.17V	132mV
	3.6V	5.05V	103mV	3.31V	102mV
	4.2V	5.03V	95mV	3.32V	98mV

On peut constater ici que, sous charge, le bruit des alimentations augmente d'un facteur 2 environ. Toutes les mesures restent dans les tolérances, avec une marge d'environ 50% pour l'alimentation 3.3V et de plus de 75% pour celle de 5V. On remarque que le bruit de l'alimentation 3.3V est relativement faible tant que le découpage ne démarre pas (il est tout de même existant car le découpage du 5V influe sur les lignes 3.3V). Lorsque la tension d'entrée passe en dessous de 3.4V, le découpage démarre et le bruit augmente d'environ 40%.

Un dernier test intéressant va permettre de valider complètement la stabilité en tension et d'assurer l'apport en courant de la ligne 5V : le branchement d'un disque-dur 2.5 pouces alimenté par le port. Ces disques sont de plus en plus répandus et très pratiques du fait qu'aucune alimentation externe n'est nécessaire. Ils tirent le courant nécessaire du port USB, ce dernier étant soumis à une forte demande lors des accès aux plateaux. Le modèle utilisé pour le test est un Western Digital WD600UO17-RTE d'une capacité de 80GB.

Mesure du 5V : sur les bornes d'un des connecteurs USB (la ligne USB étant chargée avec une souris optique et une clé mémoire USB)

Condition au repos : disque dur branché, aucun accès.

En charge : disque dur branché, copie d'un fichier de 10 Mo depuis le disque dans la mémoire Flash interne.

Voici les résultats obtenus :

Condition	Ligne 5V	
	Tension	**Bruit**
Au repos, Vbatt=4.2V	5.13V	162mVpp
En charge, VBatt=4.2V	5.10V	212mVpp
Au repos, Vbatt=3.6V	5.14V	189mVpp
En charge, VBatt=3.6V	5.11V	204mVpp
Au repos, Vbatt=2.8V	5.16V	221mVpp
En charge, VBatt=2.8V	5.13V	252mVpp

On peut déjà dire que le disque dur fonctionne parfaitement. La tension varie entre 5.10 et 5.16V ce qui est parfaitement dans les tolérances. Le bruit peut avoisiner les 250mVpp lorsque la tension batterie est très basse (2.8V). C'est beaucoup mais comparativement aux tolérances admises par la norme USB (500mVpp), c'est acceptable avec une marge de 50%.

Un test ultime a été effectué avec 2 résistances variables de puissances réglées sur une valeur de 10 ohms et connectées sur les ports USB (courant total = 1A). Le PXA320 était connecté et effectuait une boucle infinie afin de maximiser sa consommation. Voici les résultats :

Vbatt	Ligne 5V		Ligne 3.3V	
	Tension	**Bruit**	**Tension**	**Bruit**
4.2V	5.10V	131mVpp	3.31V	142mVpp
3.6V	5.08V	124mVpp	3.27V	146mVpp
2.8V	4.95V	118mVpp	3.25V	161mVpp

Remarque : Bien que, comme auparavant, les résultats soient dans les tolérances, il était intéressant de mesurer le courant tiré au niveau de la batterie (à l'ampèremètre). Celui-ci avoisine les 2.7A lors des conditions les plus défavorables (Vbatt=2.8V). Bien que la batterie permette un tel courant de décharge, il est inutile de préciser qu'elle se videra à une vitesse affolante !

Liste des composants à ajouter dans Platform Builder (1/2)

- wce500_Colibri_dotnetv2_presviewer_storageManager_usbdev Catalog items
  - Colibri: ARMV4I
    - Device Drivers
      - Display
        - PXA27x Display
      - Input Devices
        - PXA27x PS/2 Keyboard/Mouse English
      - Networking
        - Local Area Networking (LAN) devices
          - PXA27X Ethernet Davicom
      - Storage Devices
        - PXA27x Flash Strata
      - USB Function
        - USB Function Bus Drivers
          - PXA27x USB Function
      - USB Host
        - PXA27x USB Host (OHCI)
    - Ethernet Bootloader (eboot)
  - Device Drivers
    - Networking
      - Personal Area Networking (PAN) devices
        - Serial Infrared (SIR) Devices
          - Serial Infrared (SIR)
      - Wide Area Networking (WAN) devices
        - Asyncmac NDIS Driver
    - Printer Devices
    - Serial
      - Com16550
        - $(_WINCEROOT)\public\common\oak\DRIVERS\SERIAL\COM16550
        - com_mdd2
        - ser16550
      - Com_Card
        - $(_WINCEROOT)\public\common\oak\DRIVERS\SERIAL\COM_CARD
        - com_mdd2
        - ser16550
        - ser_card2
    - Storage Devices
      - Storage Devices
    - USB Function
      - USB Function Bus Drivers
        - Netchip NET2280 USB Function
      - USB Function Clients
        - Mass Storage
        - RNDIS Client
        - Serial
  - Mobile Handheld[Windows CE devices]
    - Applications - End User
      - ActiveSync
        - $(_WINCEROOT)\public\DataSync\oak\ctlpanel
        - ActiveSync system cpl
        - File Sync
      - CAB File Installer/Uninstaller
      - File Viewers
        - Microsoft Excel Viewer
        - Microsoft Image Viewer
        - Microsoft PDF Viewer
        - Microsoft PowerPoint Viewer
        - Microsoft Word Viewer
      - Help
    - Servers
      - Core Server Support
      - Simple Network Time Protocol (SNTP)
        - SNTP Client with DST
    - Core OS Services
      - Debugging Tools
        - Toolhelp API
      - Device Manager
      - Display Support
      - Internet Appliance (IABASE) Support
      - Kernel Features
        - FormatMessage API
        - Memory Mapped Files
        - Message Queue - Point-to-Point
        - Target Control Support (Shell.exe)
      - Notification LED Support
        - Notification LED APIs
      - PNP Notifications
      - Power Management (Full)
        - Common Power Manager PDD Routines
        - Power Manager MDD
        - Sample Power Manager PDD
        - Sample Power Manager PDD
      - Serial Port Support
      - UI based Notification
      - USB Host Support
        - EHCI Host Controller Driver
        - EHCI library
        - OHCI Host Controller Driver
        - OHCI Host Controller Library
        - UHCI Host Controller Driver
        - UHCI library
        - USB Driver
        - USB Human Input Device (HID) Class Driver
        - USB Printer Class Driver
        - USB Remote NDIS Class Driver
        - USB Storage Class Driver
    - File Systems and Data Store
      - Bit-based
      - Compression
      - Database Support
      - RAM and ROM File System
      - Storage Manager
        - Binary Rom Image File System
        - CD/UDFS File System
        - FAT File System
        - Landscape Storage Properties
        - Partition Driver
        - Portrait Storage Properties
        - Storage Manager Control Panel Applet
        - Transaction-Safe FAT File System (TFAT)
      - System Password
    - Fonts
      - Tahoma
        - Tahoma (Subset 1_07)
    - Graphics and Multimedia Technologies
      - Graphics
        - Gradient Fill Support
      - Imaging

Liste des composants à ajouter dans Platform Builder (2/2)

- Applications and Services Development
  - .NET Compact Framework
    - .NET Compact Framework 1.0
    - OS Dependencies for .NET Compact Framework 1.0
  - .NET Compact Framework 2.0
    - .NET Compact Framework 2.0
      - .NET Compact Framework 2.0 String Resources
        - String Resources French(France)
        - String Resources German(Germany)
        - String Resources Italian(Italy)
    - OS Dependencies for .NET Compact Framework 2.0
  - C Libraries and Runtimes
    - C++ Runtime Support for Exception Handling and Runtime Type Information
    - Full C Runtime
    - Standard I/O (STDIO)
    - Standard I/O ASCII (STDIOA)
    - Standard String Functions - ASCII (corestra)
    - String Safe Utility Functions
  - Component Services (COM and DCOM)
    - Component Object Model
      - COM
  - Object Exchange Protocol (OBEX)
    - OBEX Client
  - XML
    - MSXML 3.0
      - XML Core Services and Document Object Model (DOM)
        - XML Query Languages (XQL)
- Communication Services and Networking
  - Networking - Local Area Network (LAN)
    - Wired Local Area Network (802.3, 802.5)
    - Wireless LAN (802.11) STA - Automatic Configuration and 802.1x
  - Networking - Personal Area Network (PAN)
    - Bluetooth
      - Bluetooth Protocol Stack with Transport Driver Support
        - Bluetooth Stack with Universal Loadable Driver
    - IrDA
  - Networking - Wide Area Network (WAN)
    - Dial Up Networking (RAS/PPP)
      - AutoDial
      - Network and Dial-up Connections
      - Standard Modem Support for Dial Up Networking
    - Telephony API (TAPI 2.0)
      - Unimodem support
    - Virtual Private Networking
      - PPTP
  - Networking Features
    - Extensible Authentication Protocol
    - NDIS User-mode I/O Driver
      - NDIS user I/O utility
    - Network Driver Architecture (NDIS)
    - TCP/IP
      - IP Helper API
      - TCP/IP Checksum
      - TCP/IP Procotol
      - TCP/IP Procotol
    - TCP/IPv6 Support
    - Windows Networking API/Redirector (SMB/CIFS)
    - Winsock Support
    - Still Image Codec Support (Encode and Decode)
    - Still Image Decoders
      - BMP Decoder
      - GIF Decoder
      - JPG Decoder
      - PNG Decoder
    - Still Image Encoders
      - BMP Encoder
      - GIF Encoder
      - JPG Encoder
      - PNG Encoder
- International
  - Input Method Manager (IMM)
  - National Language Support (NLS)
- Internet Client Services
  - Browser Application
    - Pocket Internet Explorer
  - Internet Explorer 6.0 for Windows CE Components
    - Internet Explorer Browser Control Host
    - Internet Explorer Multiple-Language Base API
    - URL Moniker Services
    - Windows Internet Services
  - Internet Options Control Panel
  - Pocket Internet Explorer HTML View (WEBVIEW)
  - Scripting
    - JScript 5.6
- Security
  - Authentication Services (SSPI)
    - NTLM
    - Schannel (SSL/TLS)
  - Credential Manager
  - Cryptography Services (CryptoAPI 1.0) with High Encryption Provider
    - Certificates (CryptoAPI 2.0)
- Shell and User Interface
  - Graphics, Windowing and Events
    - Minimal GDI Configuration
    - Minimal GWES Configuration
    - Minimal Input Configuration
    - Minimal Window Manager Configuration
  - Shell
    - AYGShell API Set
    - Standard Shell
  - User Interface
    - Common Controls
      - Common Control
    - Common Dialog Support
    - Control Panel Applets
    - Mouse
    - Network User Interface
    - Overlapping Menus
    - Software Input Panel
      - Software-based Input Panel (SIP) (Choose 1 or more)
        - SIP for Small Screens
      - Software-based Input Panel Driver

regAccess.dll

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently

#if !defined(AFX_STDAFX_H__24779CA6_6FD7_44F1_8F0B_75F0F4B9F314__INCLUDED_)
#define AFX_STDAFX_H__24779CA6_6FD7_44F1_8F0B_75F0F4B9F314__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#define WIN32_LEAN_AND_MEAN          // Exclude rarely-used stuff from Windows headers

#include <windows.h>
#include <pkfuncs.h>                 // library needed to access to VirtualCopy function

#include "PXAReg.h"                  // hardware PXA270/320 registers and structure definition
#include "regAccess.h"               // header containing exported dll functions
#include "gpioFuncs.h"
#include "i2cFuncs.h"
```

RegAccess.h

```
// defined with this macro as being exported.
#ifdef REGACCESS_EXPORTS
#define REGACCESS_API __declspec(dllexport)
#else
#define REGACCESS_API __declspec(dllimport)
#endif

void* MapRegister(DWORD , DWORD );
BOOL UnMapRegister(void*);

// the following functions are shared in the DLL and are
// accessible from external programs

extern "C" _declspec(dllexport) BOOL initButtons(void);
extern "C" _declspec(dllexport) BOOL initStateLed(void);
extern "C" _declspec(dllexport) int getUpButton(void);
extern "C" _declspec(dllexport) int getDownButton(void);
extern "C" _declspec(dllexport) int getOkButton(void);
extern "C" _declspec(dllexport) int getPwrButton(void);
extern "C" _declspec(dllexport) BOOL shutdownDevice(void);
extern "C" _declspec(dllexport) BOOL setStateLed(int);

#ifdef PXA_320
       // i2c registers are not defined, disable i2c function declaration
#else
       extern "C" _declspec(dllexport) BOOL I2CWriteISL6295Reg(char, int, char, char);
       extern "C" _declspec(dllexport) int I2CReadISL6295Reg(char, int, char);
       extern "C" _declspec(dllexport) BOOL initI2CController(void);
#endif
```

RegAccess.cpp

```
#include "stdafx.h"


/*******************************************************************************/
/*      DLL MAIN DEFINITION / (EXPORTED FUNCTIONS DEFINED IN REGACCESS.H)        */
/*******************************************************************************/


// DLL creation required code
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                        )
{
    switch (ul_reason_for_call)
        {
                case DLL_PROCESS_ATTACH:
                case DLL_THREAD_ATTACH:
                case DLL_THREAD_DETACH:
                case DLL_PROCESS_DETACH:
                        break;
    }
    return TRUE;
}

/*******************************************************************************/
/*                     WIN CE HARDWARE REGISTERS ACCESS LOW LEVEL FUNCTIONS       */
/*******************************************************************************/


// Memory mapping function defined in pkfuncs.h
// (see project preprocessor C++ include directory settings)
BOOL VirtualCopy(LPVOID lpvDest, LPVOID lpvSrc, DWORD cbSize, DWORD fdwProtect);


// description :      Function which map physical memory to a virtual address space
//                                   in the memory program (here in the dll process memory)
// parameters :
//             - DWORD pageAddress    | physical page address to map
//             - DWORD pageSize       | size of the page in bytes
// return value : Pointer to void | allocated address if success, NULL if error
void* MapRegister(DWORD pageAddress, DWORD pageSize)
{
        DWORD* pRegs=NULL;

        // allocate a virtual memory space of parameter specified size in process memory
        // continue if success, return if error
        if(pRegs = (unsigned long *)VirtualAlloc(0, pageSize, MEM_RESERVE, PAGE_NOACCESS))
        {

                // map virtual reserved space with physical memory specified in parameters with
                // start address and pagesize
                if(!VirtualCopy(pRegs, (LPVOID)(pageAddress>>8), pageSize, PAGE_READWRITE |
                        PAGE_PHYSICAL |  PAGE_NOCACHE))
                {
                        // if error during virtualCopy mapping, release local process reserved
                        // memory and return a NULL pointer
                        VirtualFree(pRegs, 0, MEM_RELEASE);
                        pRegs=NULL;
                }
        }

        return pRegs;

}

// description :      Function which release local process reserved memory
// parameters :
//             - (void*)      pRegs   | pointer to reserved memory space
// return value : BOOL | non-zero if success, 0 if failure
BOOL UnMapRegister(void* pRegs)
{
        return VirtualFree(pRegs, 0, MEM_RELEASE);
}
```

GPIOFuncs.h

```
BOOL setGPIODirection(int, BOOL);
BOOL setGPIOSpecFunc(int, char);
BOOL setGPIO(int, BOOL);
BOOL getGPIO(int);
BOOL initButtons(void);
BOOL initStateLed(void);
int getUpButton();
int getDownButton();
int getOkButton();
int getPwrButton();
BOOL shutdownDevice();
BOOL setStateLed(int);
```

PXARegs.h

```
/*******************************************************************************/
/*                    PXA270/320 GENERAL REGISTERS DEFINITION                  */
/*******************************************************************************/



#ifdef MOD_PXA320

        // hardware registers base addresses definition
        #define GPIO_REGBASE          0x40E00000

        // registers page size for mapping definition
        #define GPIO_PAGE_SIZE        4096

        // GPIO general definition and configuration bits
        #define GPIO_FIRST            0
        #define GPIO_LAST             127
        #define DIR_OUTPUT            0
        #define DIR_INPUT             1
        #define FUNCTION_GPIO         0
        #define FUNCTION_AF1          1
        #define FUNCTION_AF2          2
        #define FUNCTION_AF3          3
        #define LEVEL_LOW             0
        #define LEVEL_HIGH            1

#else

        // hardware registers base addresses definition
        #define I2C_REGBASE           0x40300000
        #define CLK_REGBASE           0x41300000
        #define GPIO_REGBASE          0x40E00000

        // registers page size for mapping definition
        #define I2C_PAGE_SIZE         (4096 * 2)
        #define CLK_PAGE_SIZE         4096
        #define GPIO_PAGE_SIZE        4096

        // GPIO general definition and configuration bits
        #define GPIO_FIRST            0
        #define GPIO_LAST             120
        #define DIR_OUTPUT            0
        #define DIR_INPUT             1
        #define FUNCTION_GPIO         0
        #define FUNCTION_AF1          1
        #define FUNCTION_AF2          2
        #define FUNCTION_AF3          3
        #define LEVEL_LOW             0
        #define LEVEL_HIGH            1

        // clock registers needed configuration bits definition
        #define CKEN_I2C      (1 << 14)

        // i2c registers needed configuration bits
        #define ICR_START       (1 << 0)      // start bit
        #define ICR_STOP        (1 << 1)      // stop bit
        #define ICR_ACKNAK      (1 << 2)      // send ACK(0) or NAK(1)
        #define ICR_TB          (1 << 3)      // transfer byte bit
        #define ICR_MA          (1 << 4)      // master abort
        #define ICR_SCLE        (1 << 5)      // master clock enable
        #define ICR_IUE         (1 << 6)      // unit enable
        #define ICR_GCD         (1 << 7)      // general call disable
        #define ICR_ITEIE       (1 << 8)      // enable tx interrupts
        #define ICR_IRFIE       (1 << 9)      // enable rx interrupts
        #define ICR_BEIE        (1 << 10)     // enable bus error ints
        #define ICR_SSDIE       (1 << 11)     // slave STOP detected int enable
        #define ICR_ALDIE       (1 << 12)     // enable arbitration interrupt
        #define ICR_SADIE       (1 << 13)     // slave address detected int enable
        #define ICR_UR          (1 << 14)     // unit reset

        #define ISR_RWM         (1 << 0)      // read/write mode
        #define ISR_ACKNAK      (1 << 1)      // ack/nak status
        #define ISR_UB          (1 << 2)      // unit busy
        #define ISR_IBB         (1 << 3)      // bus busy
```

```
        #define ISR_SSD         (1 << 4)      // slave stop detected
        #define ISR_ALD         (1 << 5)      // arbitration loss detected
        #define ISR_ITE         (1 << 6)      // tx buffer empty
        #define ISR_IRF         (1 << 7)      // rx buffer full
        #define ISR_GCAD        (1 << 8)      // general call address detected
        #define ISR_SAD         (1 << 9)      // slave address detected
        #define ISR_BED         (1 << 10)     // bus error no ACK/NAK
#endif


/*******************************************************************************/
/*             PXA270/320 IPRESENTER2 HARDWARE CONFIGURATION                   */
/*******************************************************************************/

// GPIO IPRESENTER2 MAPPING CONFIGURATION DEFINITION
#ifdef MOD_PXA320
        #define GPIO_PWRBTN           53
        #define GPIO_UPBTN            77
        #define GPIO_DOWNBTN          79
        #define GPIO_OKBTN            76
        #define GPIO_PWRHOLD          94
        #define GPIO_I2CSCL           32
        #define GPIO_I2CSDA           33
        #define GPIO_GREENLED         75
        #define GPIO_REDLED           55
#else
        #define GPIO_PWRBTN           83
        #define GPIO_UPBTN            53
        #define GPIO_DOWNBTN          82
        #define GPIO_OKBTN            52
        #define GPIO_PWRHOLD          37
        #define GPIO_I2CSCL           117
        #define GPIO_I2CSDA           118
        #define GPIO_GREENLED         20
        #define GPIO_REDLED           17
#endif


// STATE LED DEFINITION
#define STATELED_OFF   0
#define STATELED_GREEN 1
#define STATELED_RED   2
#define STATELED_BOTH  3



/*******************************************************************************/
/*             PXA270/320 REGISTERS MAPPING STRUCTURES DEFINITION              */
/*******************************************************************************/

#ifdef MOD_PXA320

        // GPIO matching structure definition
        typedef volatile struct
        {
                DWORD gplr0;          // Level Reg. Bank 0
                DWORD gplr1;          // Level Reg. Bank 1
                DWORD gplr2;          // Level Reg. Bank 2
                DWORD gpdr0;          // Direction Reg. Bank 0
                DWORD gpdr1;          // Direction Reg. Bank 1
                DWORD gpdr2;          // Direction Reg. Bank 2
                DWORD gpsr0;          // Output Set Reg. Bank 0
                DWORD gpsr1;          // Output Set Reg. Bank 1
                DWORD gpsr2;          // Output Set Reg. Bank 2
                DWORD gpcr0;          // Output Clr Reg. Bank 0
                DWORD gpcr1;          // Output Clr Reg. Bank 1
                DWORD gpcr2;          // Output Clr Reg. Bank 2
                DWORD grer0;          // Ris. Edge Detect Enable Reg. Bank 0
                DWORD grer1;          // Ris. Edge Detect Enable Reg. Bank 1
                DWORD grer2;          // Ris. Edge Detect Enable Reg. Bank 2
                DWORD gfer0;          // Fal. Edge Detect Enable Reg. Bank 0
                DWORD gfer1;          // Fal. Edge Detect Enable Reg. Bank 1
                DWORD gfer2;          // Fal. Edge Detect Enable Reg. Bank 2
                DWORD gedr0;          // Edge Detect Status Reg. Bank 0
                DWORD gedr1;          // Edge Detect Status Reg. Bank 1
```

```
                DWORD gedr2;            // Edge Detect Status Reg. Bank 2
                DWORD gafr0_l;          // Alt. Function Select Reg.[  0:15 ]
                DWORD gafr0_u;          // Alt. Function Select Reg.[ 16:31 ]
                DWORD gafr1_l;          // Alt. Function Select Reg.[ 32:47 ]
                DWORD gafr1_u;          // Alt. Function Select Reg.[ 48:63 ]
                DWORD gafr2_l;          // Alt. Function Select Reg.[ 64:79 ]
                DWORD gafr2_u;          // Alt. Function Select Reg.[ 80:95 ]
                DWORD gafr3_l;          // Alt. Function Select Reg.[ 96:111]
                DWORD gafr3_u;          // Alt. Function Select Reg.[112:120]
                DWORD res1[35];
                DWORD gplr3;            // Level Detect Reg. Bank 3
                DWORD res2[2];
                DWORD gpdr3;            // Data Direction Reg. Bank 3
                DWORD res3[2];
                DWORD gpsr3;            // Pin Output Set Reg. Bank 3
                DWORD res4[2];
                DWORD gpcr3;            // Pin Output Clr Reg. Bank 3
                DWORD res5[2];
                DWORD grer3;            // Ris. Edge Detect Enable Reg. Bank 3
                DWORD res6[2];
                DWORD gfer3;            // Fal. Edge Detect Enable Reg. Bank 3
                DWORD res7[2];
                DWORD gedr3;            // Edge Detect Status Reg. Bank 3

        } GPIO_REG;

#else

        // GPIO matching structure definition
        typedef volatile struct
        {
                DWORD gplr0;            // Level Reg. Bank 0
                DWORD gplr1;            // Level Reg. Bank 1
                DWORD gplr2;            // Level Reg. Bank 2
                DWORD gpdr0;            // Direction Reg. Bank 0
                DWORD gpdr1;            // Direction Reg. Bank 1
                DWORD gpdr2;            // Direction Reg. Bank 2
                DWORD gpsr0;            // Output Set Reg. Bank 0
                DWORD gpsr1;            // Output Set Reg. Bank 1
                DWORD gpsr2;            // Output Set Reg. Bank 2
                DWORD gpcr0;            // Output Clr Reg. Bank 0
                DWORD gpcr1;            // Output Clr Reg. Bank 1
                DWORD gpcr2;            // Output Clr Reg. Bank 2
                DWORD grer0;            // Ris. Edge Detect Enable Reg. Bank 0
                DWORD grer1;            // Ris. Edge Detect Enable Reg. Bank 1
                DWORD grer2;            // Ris. Edge Detect Enable Reg. Bank 2
                DWORD gfer0;            // Fal. Edge Detect Enable Reg. Bank 0
                DWORD gfer1;            // Fal. Edge Detect Enable Reg. Bank 1
                DWORD gfer2;            // Fal. Edge Detect Enable Reg. Bank 2
                DWORD gedr0;            // Edge Detect Status Reg. Bank 0
                DWORD gedr1;            // Edge Detect Status Reg. Bank 1
                DWORD gedr2;            // Edge Detect Status Reg. Bank 2
                DWORD gafr0_l;          // Alt. Function Select Reg.[  0:15 ]
                DWORD gafr0_u;          // Alt. Function Select Reg.[ 16:31 ]
                DWORD gafr1_l;          // Alt. Function Select Reg.[ 32:47 ]
                DWORD gafr1_u;          // Alt. Function Select Reg.[ 48:63 ]
                DWORD gafr2_l;          // Alt. Function Select Reg.[ 64:79 ]
                DWORD gafr2_u;          // Alt. Function Select Reg.[ 80:95 ]
                DWORD gafr3_l;          // Alt. Function Select Reg.[ 96:111]
                DWORD gafr3_u;          // Alt. Function Select Reg.[112:120]
                DWORD res1[35];
                DWORD gplr3;            // Level Detect Reg. Bank 3
                DWORD res2[2];
                DWORD gpdr3;            // Data Direction Reg. Bank 3
                DWORD res3[2];
                DWORD gpsr3;            // Pin Output Set Reg. Bank 3
                DWORD res4[2];
                DWORD gpcr3;            // Pin Output Clr Reg. Bank 3
                DWORD res5[2];
                DWORD grer3;            // Ris. Edge Detect Enable Reg. Bank 3
                DWORD res6[2];
                DWORD gfer3;            // Fal. Edge Detect Enable Reg. Bank 3
                DWORD res7[2];
                DWORD gedr3;            // Edge Detect Status Reg. Bank 3
```

```
        } GPIO_REG;

        // I2C matching structure definition
        typedef volatile struct
        {
                char PADING[0x167F];
                DWORD IBMR;            // I2C Bus Monitor Register - IBMR
                DWORD RESERVED1;
                DWORD IDBR;            // I2C Data Buffer Register - IDBR
                DWORD RESERVED2;
                DWORD ICR;             // I2C Control Register - ICR
                DWORD RESERVED3;
                DWORD ISR;             // I2C Status Register - ISR
                DWORD RESERVED4;
                DWORD ISAR;            // I2C Slave Address Register - ISAR
                DWORD RESERVED5;

        } I2C_REG;


        // CLK matching structure definition
        typedef volatile struct
        {
                DWORD CCCR;             // Core Clock Configuration Register
                DWORD CKEN;             // Clock Enable Register
                DWORD OSCC;             // Oscillator Configuration Register
                DWORD CCSRM;            // Core Clock Status Register


        } CLK_REG;

#endif
```

GPIOFuncs.cpp

```
#include "StdAfx.h"


/*****************************************************************************/
/*                            PXA270/320 GPIO REGISTERS LOW LEVEL FUNCTIONS
        */
/*****************************************************************************/


// description : Function wich set direction of a GPIO
// parameters :
//             - int   gpioN          | GPIO id to modify | number between 0 and 120
//             - BOOL  direction      | port direction | FALSE = output, TRUE = input
// return value : BOOL        | non-zero if success, 0 if failure
BOOL setGPIODirection(int gpioN, BOOL direction)
{
        // Control validity of given parameters
        if((gpioN >= GPIO_FIRST) & (gpioN <= GPIO_LAST))
        {

                // GPIO memory registers structure type
                GPIO_REG*      pGPIO_REGs;

                // try to map local structure to physical memory (GPIO Controller area)
                pGPIO_REGs=(volatile GPIO_REG *)MapRegister(GPIO_REGBASE, GPIO_PAGE_SIZE);

                // check if the mapping is done
                if(pGPIO_REGs != NULL)
                {

                        if(gpioN<32)
                        {
                                if(direction==DIR_INPUT)      pGPIO_REGs->gpdr0 &= ~(1 << gpioN);
                                else                          pGPIO_REGs->gpdr0 |=  (1 << gpioN);
                        }
                        else if(gpioN<64)
                        {
                                if(direction==DIR_INPUT)      pGPIO_REGs->gpdr1 &= ~(1 << (gpioN-32));
                                else                          pGPIO_REGs->gpdr1 |=  (1 << (gpioN-32));
                        }
                        else if(gpioN<96)
                        {
                                if(direction==DIR_INPUT)      pGPIO_REGs->gpdr2 &= ~(1 << (gpioN-64));
                                else                          pGPIO_REGs->gpdr2 |=  (1 << (gpioN-64));
                        }
                        else
                        {
                                if(direction==DIR_INPUT)      pGPIO_REGs->gpdr3 &= ~(1 << (gpioN-96));
                                else                          pGPIO_REGs->gpdr3 |=  (1 << (gpioN-96));
                        }


                        // release all bindings to physical memory
                        UnMapRegister((void*)pGPIO_REGs);

                        return TRUE;

                }
                return FALSE;
        }
        return FALSE;
}
```

```
// Function wich set function of a GPIO, return FALSE if success
// parameters :
//              - int   gpioN          | GPIO id to modify    | number between 0 and 120
//              - char  selFunc        | port function        | 0 = I/O, 1,2,3 = special function
// return value : BOOL        | non-zero if success, 0 if failure
BOOL setGPIOSpecFunc(int gpioN, char selFunc)
{

        // Control validity of given parameters
        if( (gpioN >= GPIO_FIRST) & (gpioN <= GPIO_LAST) &
                ((selFunc == FUNCTION_GPIO)   |
                (selFunc == FUNCTION_AF1)     |
                (selFunc == FUNCTION_AF2)     |
                (selFunc == FUNCTION_AF3)))
        {

                // GPIO memory registers structure type
                GPIO_REG*      pGPIO_REGs;

                // try to map local structure to physical memory (GPIO Controller area)
                pGPIO_REGs=(volatile GPIO_REG *)MapRegister(GPIO_REGBASE, GPIO_PAGE_SIZE);

                // check if the mapping is done
                if(pGPIO_REGs != NULL)
                {
                        // modify corresponding register with parameter given value
                        if      (gpioN<16)     pGPIO_REGs->gafr0_l |= (selFunc << (gpioN*2));
                        else if (gpioN<32)     pGPIO_REGs->gafr0_u |= (selFunc << ((16-gpioN)*2));
                        else if (gpioN<48)     pGPIO_REGs->gafr1_l |= (selFunc << ((32-gpioN)*2));
                        else if (gpioN<64)     pGPIO_REGs->gafr1_u |= (selFunc << ((48-gpioN)*2));
                        else if (gpioN<82)     pGPIO_REGs->gafr2_l |= (selFunc << ((64-gpioN)*2));
                        else if (gpioN<96)     pGPIO_REGs->gafr2_u |= (selFunc << ((82-gpioN)*2));
                        else if (gpioN<112)    pGPIO_REGs->gafr3_l |= (selFunc << ((96-gpioN)*2));
                        else                   pGPIO_REGs->gafr3_u |= (selFunc << ((112-gpioN)*2));

                        // release all bindings to physical memory
                        UnMapRegister((void*)pGPIO_REGs);

                        return TRUE;
                }
                else
                {
                        return FALSE;
                }
        }
        else
        {
                return FALSE;
        }
}


// Function wich set level for a GPIO
// parameters :
//              - int   gpioN          | GPIO id to modify | number between 0 and 120
//              - char  selFunc        | port function | 0 = I/O, 1,2,3 = special function
// return value : BOOL        | non-zero if success, 0 if failure
BOOL setGPIO(int gpioN, BOOL value)
{
        // Control validity of given parameters
        if((gpioN >= GPIO_FIRST) & (gpioN <= GPIO_LAST))
        {

                // GPIO memory registers structure type
                GPIO_REG*      pGPIO_REGs;

                // try to map local structure to physical memory (GPIO Controller area)
                pGPIO_REGs=(volatile GPIO_REG *)MapRegister(GPIO_REGBASE, GPIO_PAGE_SIZE);


                // check if the mapping is done
                if(pGPIO_REGs != NULL)
                {
```

```
                        // GPIO to set is in bank 0 (GPIO0...GPIO31)
                        if(gpioN<32)
                        {
                                if(value==LEVEL_LOW)  pGPIO_REGs->gpcr0=(1 << gpioN);// Set GPIOxx val at 0
                                else                  pGPIO_REGs->gpsr0=(1 << gpioN);// Set GPIOxx val at 1
                        }

                        // GPIO to set is in bank 1 (GPIO32...GPIO63)
                        else if(gpioN < 64)
                        {
                                if(value==LEVEL_LOW)  pGPIO_REGs->gpcr1=(1 << (gpioN-32));// Set GPIOxx at 0
                                else                  pGPIO_REGs->gpsr1=(1 << (gpioN-32));// Set GPIOxx at 1
                        }

                        // GPIO to set is in bank 2 (GPIO64...GPIO95)
                        else if(gpioN < 96)
                        {
                                if(value==LEVEL_LOW)  pGPIO_REGs->gpcr2=(1 << (gpioN-64));// Set GPIOxx at 0
                                else                  pGPIO_REGs->gpsr2=(1 << (gpioN-64));// Set GPIOxx at 1
                        }

                        // GPIO to set is in bank 3 (GPIO96...GPIO119)
                        else
                        {
                                if(value==LEVEL_LOW)  pGPIO_REGs->gpcr3=(1 << (gpioN-96));// Set GPIOxx at 0
                                else                  pGPIO_REGs->gpsr3=(1 << (gpioN-96));// Set GPIOxx at 1
                        }

                        // release all bindings to physical memory
                        UnMapRegister((void*)pGPIO_REGs);

                        return TRUE;
                }
                return FALSE;
        }
        return FALSE;
}

// Function wich get level for a GPIO
// parameters :
//              - int   gpioN | GPIO id to modify | number between 0 and 120
// return value : int | -1 if error, pin value if success (0 or 1)
int getGPIO(int gpioN)
{

        // variable used as return value
        int retVal=-1;

        // Control validity of given parameters
        if((gpioN >= GPIO_FIRST) & (gpioN <= GPIO_LAST))
        {

                // GPIO memory registers structure type
                GPIO_REG*       pGPIO_REGs;

                // try to map local structure to physical memory (GPIO Controller area)
                pGPIO_REGs=(volatile GPIO_REG *)MapRegister(GPIO_REGBASE, GPIO_PAGE_SIZE);

                // check if the mapping is done
                if(pGPIO_REGs != NULL)
                {

                        // GPIO to get is in bank 0 (GPIO0...GPIO31)
                        if(gpioN<32)
                        {
                                if(pGPIO_REGs->gplr0 & (1 << gpioN))        retVal=1;// Get value of GPIOxx
                                else                                        retVal=0;
                        }

                        // GPIO to get is in bank 1 (GPIO32...GPIO63)
                        else if(gpioN < 64)
                        {
                                if(pGPIO_REGs->gplr1 & (1 << (gpioN-32)))   retVal=1;// Get value of GPIOxx
                                else                                        retVal=0;
                        }
```

```
                        // GPIO to get is in bank 2 (GPIO64...GPIO95)
                        else if(gpioN < 96)
                        {
                                if(pGPIO_REGs->gplr2 & (1 << (gpioN-64)))   retVal=1;// Get value of GPIOxx
                                else                                        retVal=0;
                        }

                        // GPIO to get is in bank 3 (GPIO96...GPIO119)
                        else if(gpioN < 120)
                        {
                                if(pGPIO_REGs->gplr3 & (1 << (gpioN-96)))   retVal=1;// Get value of GPIOxx
                                else                                        retVal=0;
                        }

                        // release all bindings to physical memory
                        UnMapRegister((void*)pGPIO_REGs);
                }
        }

        return retVal;
}


/*****************************************************************************/
/*                      IPRESENTER2 GPIO HIGH LEVEL SPECIFIC FUNCTIONS            */
/*****************************************************************************/

// function wich initializes all buttons in input mode / GPIO function mode
// return value : BOOL | FALSE if success, TRUE if error
BOOL initButtons(void)
{
        BOOL success=TRUE;    // error detection variable

        if(!setGPIODirection(GPIO_PWRBTN,    DIR_INPUT))    success=FALSE;
        if(!setGPIODirection(GPIO_UPBTN,     DIR_INPUT))    success=FALSE;
        if(!setGPIODirection(GPIO_DOWNBTN,   DIR_INPUT))    success=FALSE;
        if(!setGPIODirection(GPIO_OKBTN,     DIR_INPUT))    success=FALSE;

        if(!setGPIOSpecFunc(GPIO_PWRBTN,     FUNCTION_GPIO)) success=FALSE;
        if(!setGPIOSpecFunc(GPIO_UPBTN,      FUNCTION_GPIO)) success=FALSE;
        if(!setGPIOSpecFunc(GPIO_DOWNBTN,    FUNCTION_GPIO)) success=FALSE;
        if(!setGPIOSpecFunc(GPIO_OKBTN,      FUNCTION_GPIO)) success=FALSE;

        return success;
}


// function wich initializes the board state leds as output and off
// return value : BOOL | FALSE if success, TRUE if error
BOOL initStateLed(void)
{
        BOOL success=TRUE;    // error detection variable

        if(!setGPIODirection(GPIO_GREENLED,  DIR_OUTPUT)) success=FALSE;
        if(!setGPIODirection(GPIO_REDLED,    DIR_OUTPUT)) success=FALSE;

        if(!setGPIO(GPIO_GREENLED,    LEVEL_LOW))     success=FALSE;
        if(!setGPIO(GPIO_REDLED,      LEVEL_LOW))     success=FALSE;

        return success;
}

// function wich return value of GPIO connected with UP button
// return value : int | -1 if error, pin value if success (0 or 1)
int getUpButton()
{
        return getGPIO(GPIO_UPBTN);
}

// function wich return value of GPIO connected with DOWN button
// return value : int | -1 if error, pin value if success (0 or 1)
int getDownButton()
```

```
{
	return getGPIO(GPIO_DOWNBTN);
}

// function wich return value of GPIO connected with OK button
// return value : int | -1 if error, pin value if success (0 or 1)
int getOkButton()
{
	return getGPIO(GPIO_OKBTN);
}

// function wich return value of GPIO connected with PWR button
// return value : int | -1 if error, pin value if success (0 or 1)
int getPwrButton()
{
	return getGPIO(GPIO_PWRBTN);
}

// function wich stop holding device power and shutdown it
// return value : BOOL | FALSE if success, TRUE if error
BOOL shutdownDevice()
{
	BOOL success=true;
	if(!setGPIODirection(GPIO_PWRHOLD, DIR_OUTPUT)) success = false;
	if(!setGPIO(GPIO_PWRHOLD, LEVEL_LOW))		success = false;
	return success;
}

// function wich set leds GPIO to the 4 existant possibilites
BOOL setStateLed(int state)
{
	BOOL success=TRUE;	// error detection variable

	// it is not necessary to check parameter with a switch sequence
	switch(state)
	{
		case STATELED_OFF:
		{
			if(setGPIO(GPIO_GREENLED,	LEVEL_HIGH)) success=FALSE;
			if(setGPIO(GPIO_REDLED,		LEVEL_HIGH)) success=FALSE;
			break;
		}
		case STATELED_GREEN:
		{
			if(setGPIO(GPIO_GREENLED,	LEVEL_LOW))  success=FALSE;
			if(setGPIO(GPIO_REDLED,		LEVEL_HIGH)) success=FALSE;
			break;
		}
		case STATELED_RED:
		{
			if(setGPIO(GPIO_GREENLED,	LEVEL_HIGH)) success=FALSE;
			if(setGPIO(GPIO_REDLED,		LEVEL_LOW))  success=FALSE;
			break;
		}
		case STATELED_BOTH:
		{
			if(setGPIO(GPIO_GREENLED,	LEVEL_LOW))  success=FALSE;
			if(setGPIO(GPIO_REDLED,		LEVEL_LOW))  success=FALSE;
			break;
		}
		default :
		{
			success=TRUE;
			break;
		}
	}

	return success;
}
```

I2CFuncs.h

```
#ifdef PXA_320
        // i2c registers are not defined, disable i2c function declaration
#else
        BOOL initI2CController();
        BOOL I2CWriteISL6295Reg(char, int, char, char);
        int I2CReadISL6295Reg(char, int, char);
#endif
```

I2CFuncs.cpp

```
// General description
// functions : read / write byte on the I2C port
// always CPU Master, no arbitration
// active waiting : register polling until packet sent

#include "StdAfx.h"

/*******************************************************************************/
/*                    PXA270 I2C REGISTERS LOW LEVEL FUNCTIONS                 */
/*******************************************************************************/

#ifdef MOD_PXA320
	// i2c registers are not defined, disable i2c code
#else
	// Global variable used to set a maximum time to wait
	// during an I2C data byte is sending or receiving
	int busTimeout = 50000;

	// description : Function which waits the end of byte receiving
	// return value : BOOL | non-zero if success, 0 if failure
	BOOL I2CRxFull(I2C_REG* pI2CRegs)
	{
		// variable used to temporary store ISR register value
		DWORD temp;

		// loop variable
		int i=0;

		// Wait until ISR bit 7 is set (byte received) or timeout
		while (i<=busTimeout)
		{
			//get and store value of ISR register
			temp = pI2CRegs->ISR;

			// test if bit 7 (IRF / byte received) is set in status register
			if ((temp & (1 << 7)) == (1 << 7))
			{
				// clear bit 7 (IRF / clear interrupt) in status register
				pI2CRegs->ISR = temp | (1 << 7);
				return TRUE;
			}
			i++;
		}
		return FALSE;
	}

	// description : Function which waits the end of byte sending
	// return value : BOOL | non-zero if success, 0 if failure
	BOOL I2CTxEmpty(I2C_REG* pI2CRegs)
	{
		// variable used to temporary store ISR register value
		DWORD temp;

		// loop variable
		int i=0;

		// Wait until ISR bit 6 is set (byte sent) or timeout
		while (i<=busTimeout)
		{
			//get and store value of ISR register
			temp = pI2CRegs->ISR;

			// test if bit 6 (ITE / byte sent) is set in status register
			if((temp & (1 << 6)) == (1 << 6))
			{
				// clear bit 6 (IRF / clear interrupt) in status register
				pI2CRegs->ISR = temp | (1 << 6);
				return TRUE;
			}
			i++;
		}
		return FALSE;
	}
```

```
        // description : Function which init I2C controller unit of PXA270
        // return value : BOOL | non-zero if success, 0 if failure
        BOOL initI2CController()
        {

                // pointer on physical memory compatible structures
                static I2C_REG *pI2CRegs;            // I2C memory registers structure type
                static CLK_REG *pCLKRegs;            // CLOCK memory registers structure type
                static GPIO_REG *pGPIO_REGs;         // GPIO memory registers structure type

                // try to map local structure to physical memory (I2C Controller area)
                pI2CRegs       = (volatile I2C_REG*) MapRegister(I2C_REGBASE,      I2C_PAGE_SIZE);

                // try to map local structure to physical memory (Clocks control area)
                pCLKRegs       = (volatile CLK_REG*) MapRegister(CLK_REGBASE,      CLK_PAGE_SIZE);

                // try to map local structure to physical memory (GPIO Controller area)
                pGPIO_REGs     = (volatile GPIO_REG*) MapRegister(GPIO_REGBASE,    GPIO_PAGE_SIZE);

                // verify that all mappings are done
                if((pI2CRegs!=NULL) & (pCLKRegs!=NULL) & (pGPIO_REGs!=NULL))
                {

                        // clear controler register (ICR)
                        pI2CRegs->ICR = 0;

                        // enable I2C master clock of clocks enable control register (CKEN)
                        pCLKRegs->CKEN |=  CKEN_I2C;

                        // set SDA and SCL pin as input
                        setGPIODirection(GPIO_I2CSCL, DIR_INPUT);
                        setGPIODirection(GPIO_I2CSDA, DIR_INPUT);

                        // enable alternative function 1 for SDA and SCL pin
                        setGPIOSpecFunc(GPIO_I2CSCL, FUNCTION_AF1);
                        setGPIOSpecFunc(GPIO_I2CSDA, FUNCTION_AF1);
                        //pGPIO_REGs->gafr3_u |=  (1 << 10) | (1 << 12);

                        // Set clock master output (SCLE) and enable I2C interface (IUE)
                        // in control register (ICR)
                        pI2CRegs->ICR = ICR_SCLE;
                        pI2CRegs->ICR |= ICR_IUE;

                        // release all bindings to physical memory
                        UnMapRegister((void*)pI2CRegs);
                        UnMapRegister((void*)pCLKRegs);
                        UnMapRegister((void*)pGPIO_REGs);

                        return TRUE;
                }

                return FALSE;
        }


#endif
```

```
/*****************************************************************************/
/*            IPRESENTER2 HIGH LEVEL SPECIFIC I2C FUNCTIONS                  */
/*****************************************************************************/


#ifdef MOD_PXA320
        // i2c registers are not defined, disable i2c code
#else

        // description : Function which write a byte on a slave of the I2C bus
        // parameters :
        //              - char  slaveAddr      | slave device address| number between 0 and 255
        //              - BOOL  bank           | bank number selection       | 0=EEPROM or 1=RAM
        //              - char  regAddr        | register address            | number between 0 and 255
        //              - char  value          | value to write in reg | number between 0 and 255
        // return value : BOOL        | non-zero if success, 0 if failure
        BOOL I2CWriteISL6295Reg(char slaveAddr, BOOL bank, char regAddr, char value)
        {

                char confByte = 0;            // variable used to create config. byte to send
                static I2C_REG *pI2CRegs;     // pointer on an I2C physical memory compatible structure
                DWORD reg;                    // variable used to temporary store ICR register value
                BOOL noError=FALSE;           // error detection variable

                // try to map local structure to physical memory (I2C Controller area)
                pI2CRegs       = (volatile I2C_REG*) MapRegister(I2C_REGBASE,      I2C_PAGE_SIZE);

                // check if the mapping is done
                if(pI2CRegs != NULL)
                {

                        // place slave address with write last bit selection in data register IDBR
                        pI2CRegs->IDBR = (slaveAddr << 1) & ~(1 << 0);
                        // save control register value to local variable
                        reg = pI2CRegs->ICR;
                        // active start and transfert byte options
                        // and disable stop option
                        reg |= (ICR_START | ICR_TB);
                        reg &= ~(ICR_STOP);
                        // update ICR with selected options => the I2C unit start to send datas
                        pI2CRegs->ICR = reg;

                        // wait sending end and if no timeout, continue the procedure
                        if(I2CTxEmpty(pI2CRegs))
                        {
                                // create config byte to send and place it in data register IDBR
                                confByte |= (bank << 2);
                                pI2CRegs->IDBR = (DWORD)(confByte);
                                // save control register value to local variable
                                reg = pI2CRegs->ICR;
                                // active transfert byte option
                                // and disable start and stop options
                                reg &= ~(ICR_START | ICR_STOP);
                                reg |= (ICR_TB);
                                // update ICR with selected options => the I2C unit start to send datas
                                pI2CRegs->ICR = reg;
                                // wait sending end and if no timeout, continue the procedure
                                if(I2CTxEmpty(pI2CRegs))
                                {
                                        // Convert slave register address parameter to 32 bits and place
                                        // it in data reg (IDBR)
                                        pI2CRegs->IDBR = (DWORD)(regAddr);
                                        reg = pI2CRegs->ICR;
                                        // active transfert byte option
                                        // and disable start and stop options
                                        reg &= ~(ICR_START | ICR_STOP);
                                        reg |= (ICR_TB);
                                        // update ICR with sel. options => the I2C unit start to send datas
                                        pI2CRegs->ICR = reg;
                                        // wait sending end and if no timeout, continue the procedure
                                        if(I2CTxEmpty(pI2CRegs))
                                        {
                                                // Convert value to write in register to 32 bits and place
```

```
                                        // it in data register (IDBR)
                                        pI2CRegs->IDBR = (DWORD)(value);
                                        reg = pI2CRegs->ICR;
                                        // active transfert byte and stop option
                                        // and disable start option
                                        reg &= ~ICR_START;
                                        reg |= (ICR_STOP | ICR_TB);
                                        // update ICR with sel. options=> I2C unit start to send datas
                                        pI2CRegs->ICR = reg;
                                        // wait sending end and if no timeout, write is successed
                                        if(I2CTxEmpty(pI2CRegs))
                                        {
                                                noError=TRUE;
                                        }
                                        // Clear stop bit in control register (needed)
                                        pI2CRegs->ICR &= ~ICR_STOP;
                                }
                        }
                }

                // release all bindings to physical memory
                UnMapRegister((void*)pI2CRegs);
        }

        return noError;
}


// description : Function which read a byte from a slave of the I2C bus
// parameters :
//              - char  slaveAddr       | slave device address| number between 0 and 255
//              - BOOL  bank            | bank number selection       | 0=EEPROM or 1=RAM
//              - char  regAddr         | register address            | number between 0 and 255
// return value : int         | 0 to 255 if success, -1 if error
int I2CReadISL6295Reg(char slaveAddr, int bank, char regAddr)
{


        char confByte = 0;              // variable used to create config. byte to send
        static I2C_REG *pI2CRegs;       // pointer on an I2C physical memory compatible structure
        DWORD reg;                      // variable used to temporary store ICR register value
        int retReg=-1;                  // variable used to store register read value

        // try to map local structure to physical memory (I2C Controller area)
        pI2CRegs       = (volatile I2C_REG*) MapRegister(I2C_REGBASE, I2C_PAGE_SIZE);

        // check if the mapping is done
        if(pI2CRegs != NULL)
        {

                // place slave address with write last bit selection in data register IDBR
                pI2CRegs->IDBR = (slaveAddr << 1) & ~(1 << 0);
                // save control register value to local variable
                reg = pI2CRegs->ICR;
                // active start and transfert byte options
                // and disable stop option
                reg |= (ICR_START | ICR_TB);
                reg &= ~(ICR_STOP);
                // update ICR with selected options => the I2C unit start to send datas
                pI2CRegs->ICR = reg;
                // wait sending end and if no timeout, continue the procedure
                if(I2CTxEmpty(pI2CRegs))
                {
                        // create config byte to send and place it in data register IDBR
                        confByte |= (bank << 2);
                        pI2CRegs->IDBR = (DWORD)(confByte);
                        // save control register value to local variable
                        reg = pI2CRegs->ICR;
                        // active transfert byte option
                        // and disable start and stop options
                        reg &= ~(ICR_START | ICR_STOP);
                        reg |= (ICR_TB);
                        // update ICR with selected options => the I2C unit start to send datas
                        pI2CRegs->ICR = reg;
                        // wait sending end and if no timeout, continue the procedure
```

```
                        if(I2CTxEmpty(pI2CRegs))
                        {
                                // Convert slave register address parameter to 32 bits and place
                                // it in data reg (IDBR)
                                pI2CRegs->IDBR = (DWORD)(regAddr);
                                reg = pI2CRegs->ICR;
                                // active transfert byte option
                                // and disable start and stop options
                                reg &= ~(ICR_START | ICR_STOP);
                                reg |= (ICR_ALDIE | ICR_TB);
                                // update ICR with sel. options => the I2C unit start to send datas
                                pI2CRegs->ICR = reg;
                                // wait sending end and if no timeout, continue the procedure
                                if(I2CTxEmpty(pI2CRegs))
                                {
                                        // place slave address with read last bit selection in data
                                        // register IDBR
                                        pI2CRegs->IDBR = (slaveAddr << 1) | (1 << 0);
                                        reg = pI2CRegs->ICR;
                                        // active start (repeated start) and byte transfert options
                                        // disable stop bit and loosing arbitration options
                                        reg |= (ICR_START | ICR_TB);
                                        reg &= ~(ICR_STOP | ICR_ALDIE);
                                        // update ICR with sel. Opt. => the I2C unit start send datas
                                        pI2CRegs->ICR = reg;
                                        // wait sending end and if no timeout, continue the procedure
                                        if(I2CTxEmpty(pI2CRegs))
                                        {
                                                reg = pI2CRegs->ICR;
                                                // disable start bit option
                                                reg &= ~ICR_START;
                                                // enable byte transfert, stop bit, loosing
                                                // arbitration and  negative ack after reading options
                                                reg |= (ICR_ALDIE | ICR_TB | ICR_STOP | ICR_ACKNAK);
                                                pI2CRegs->ICR = reg;
                                                // wait recv. end and if no timeout,continue procedure
                                                if(I2CRxFull(pI2CRegs))
                                                {
                                                // take 8 firts bits of the data register (received
                                                // values) and place them on the return value variable
                                                        retReg = (char)pI2CRegs->IDBR & 0xFF;
                                                }
                                                // clear stop and negative ack bit in de control reg
                                                pI2CRegs->ICR &= ~(ICR_STOP | ICR_ACKNAK);
                                        }
                                }
                        }
                }

                // release all bindings to physical memory
                UnMapRegister((void*)pI2CRegs);
        }

        return retReg;
}

#endif
```

WceFunc.dll

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__0435AD2D_12DD_45D2_9511_22DB9809E6A8__INCLUDED_)
#define AFX_STDAFX_H__0435AD2D_12DD_45D2_9511_22DB9809E6A8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


// Insert your headers here
#define WIN32_LEAN_AND_MEAN   // Exclude rarely-used stuff from Windows headers

#include <windows.h>
#include <pkfuncs.h>          // library needed to access to KernelIoControl function
```

WceFn.h

```
#ifdef WCEFN_EXPORTS
#define WCEFN_API __declspec(dllexport)
#else
#define WCEFN_API __declspec(dllimport)
#endif

// the following functions are shared in the DLL and are
// accessible from external programs

extern "C" _declspec(dllexport) void mouse_RightClic(void);
extern "C" _declspec(dllexport) void mouse_LeftClic(void);
extern "C" _declspec(dllexport) void mouse_showcursor(BOOL curstate);
extern "C" _declspec(dllexport) void mouse_setPosition(int x, int y);
extern "C" _declspec(dllexport) void keyboard_P(void);
extern "C" _declspec(dllexport) void kernel_restart(void);
extern "C" _declspec(dllexport) BOOL setDisplayResolution(int resolution);
extern "C" _declspec(dllexport) int  getDisplayResolution();
```

WceFn.cpp

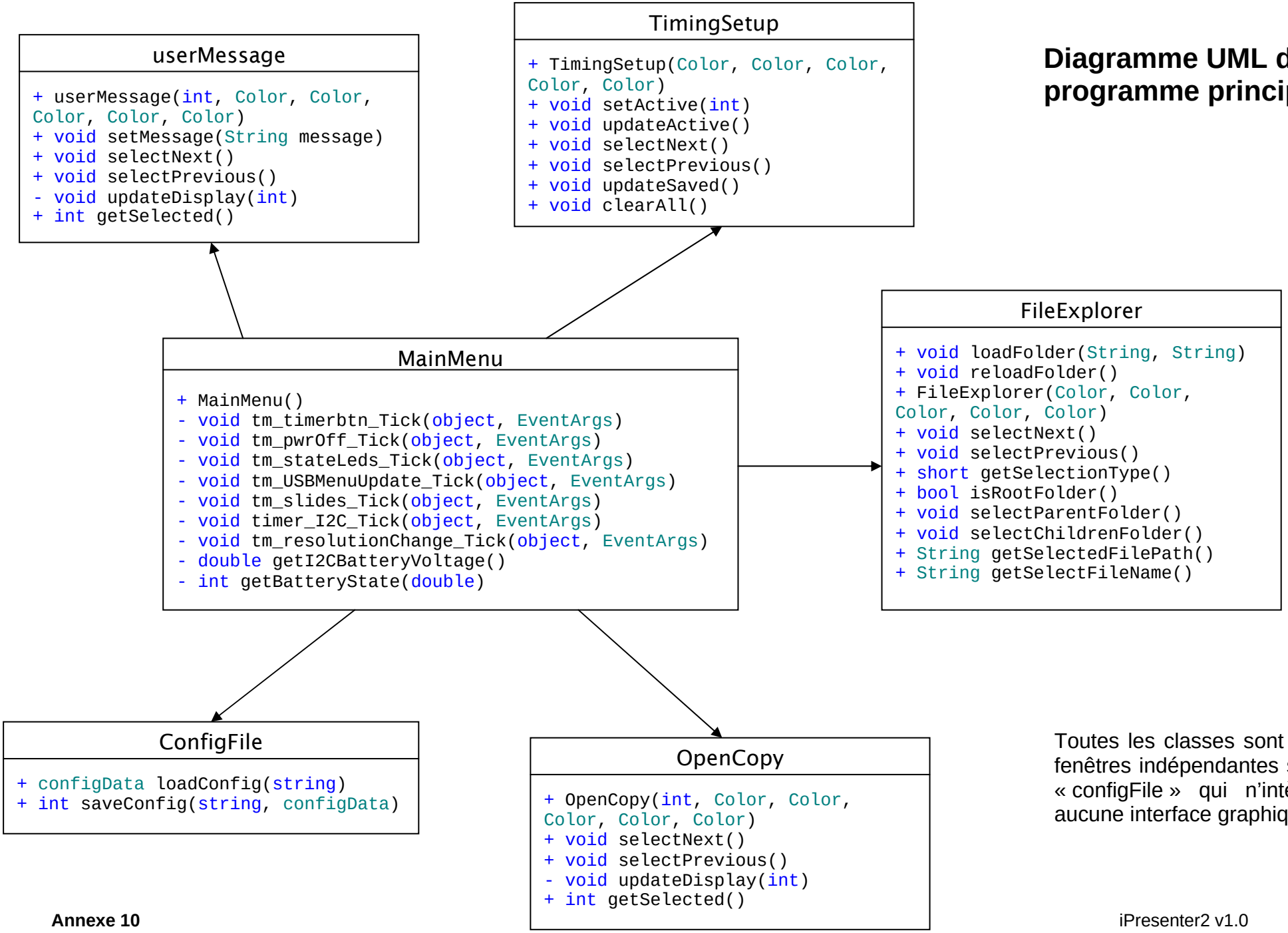
```
#include "stdafx.h"
#include "WceFn.h"

/*****************************************************************************************************************/
/*          DLL MAIN DEFINITION / (EXPORTED FUNCTIONS DEFINED IN REGACCESS.H)                                    */
/*****************************************************************************************************************/


// DLL creation required code
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved )
{
   switch (ul_reason_for_call)
            {
                        case DLL_PROCESS_ATTACH:
                        case DLL_THREAD_ATTACH:
                        case DLL_THREAD_DETACH:
                        case DLL_PROCESS_DETACH:
                        break;
   }
   return TRUE;
}



/*****************************************************************************************************************/
/*                                IPRESENTER2 WINCE5 HIGH LEVEL SPECIFIC FUNCTIONS                                */
/*****************************************************************************************************************/

// description : Function which simulate a right mouse clic
void mouse_RightClic(void)
{
            mouse_event(MOUSEEVENTF_RIGHTDOWN,    0, 0, 0, 0);
            mouse_event(MOUSEEVENTF_RIGHTUP,      0, 0, 0, 0);
}

// description : Function which simulate a left mouse clic
void mouse_LeftClic(void)
{
            mouse_event(MOUSEEVENTF_LEFTDOWN,     0, 0, 0, 0);
            mouse_event(MOUSEEVENTF_LEFTUP,       0, 0, 0, 0);
}

// description : Function which display or hide mouse cursor
//               on the desktop
// - BOOL curstate  | display cursor var | TRUE=show cursor, FALSE=hide
void mouse_showcursor(BOOL curstate)
{
            ShowCursor(curstate);
}

// description : Function which set the cursor position to parameters
//               given by x (horizontal) and y (vertical)
/ /- int     x          | horizontal position          | 0 to 1024 (at max resolution)
// - int     y          | vertical position            | 0 to 768 (at max resolution)
void mouse_setPosition(int x, int y)
{
            if((x >= 0) & (x<=1024))
            {
                        SetCursorPos(x, y);
            }
}

// description : Function which simulate a key press (capital "P")
void keyboard_P(void)
{
            keybd_event (0x50, 0x19,      KEYEVENTF_EXTENDEDKEY | 0, 0 );
             keybd_event(0x50, 0x19,      KEYEVENTF_EXTENDEDKEY | KEYEVENTF_KEYUP, 0);
}
```

```
// description : Function which write a value in a parameter's given register key
//                  - int           mainKey         | main key selection        | 0=HKLM, 1=HKU, 2=HKCR
//                  - LPCTSTR       key             | key path string           | string
//                  - LPCTSTR       valueName       | value name                | string
//                  - DWORD         value           | value content             | 0 to 18446744073709551616
// return value :   BOOL                            | TRUE if success, FALSE if error
BOOL writeRegister(int mainKey, LPCTSTR key, LPCTSTR valueName, DWORD value)
{

        struct HKEY__ * mainKeyAddr;        // main key selection handle
        struct HKEY__ * keyAddr;            // current key handle
        int statusHandler;                  // register functions return codes
        BOOL success = FALSE;               // error detection

        // replace user mainKey choice by correct handle
        switch(mainKey)
        {
                case 0 :    mainKeyAddr = HKEY_LOCAL_MACHINE;       break;
                case 1 :    mainKeyAddr = HKEY_CURRENT_USER;        break;
                case 2 :    mainKeyAddr = HKEY_CLASSES_ROOT;        break;
                default :   mainKey=-1;                             break;
        }

        // if valid main key selection
        if(mainKey!=-1)
        {

                // open selected register key and return handle to it (keyAddr)
                // in read/write mode (KEY_SET_VALUE)
                statusHandler = RegOpenKeyEx (mainKeyAddr, key, 0, KEY_SET_VALUE, &keyAddr);

                // if key found and opened
                if(statusHandler == ERROR_SUCCESS)
                {
                        // modify parameter's given value using previous handle
                        statusHandler = RegSetValueEx (keyAddr, valueName, 0,
                                                        REG_DWORD, (LPBYTE)&value,  sizeof(value));

                        // control if modification is successed
                        if(statusHandler == ERROR_SUCCESS)
                        {
                                success = TRUE;
                        }

                        // close register key
                        RegCloseKey(keyAddr);
                }
        }
        return success;
}


// description : Function which read a DWORD value in a parameter's given register key
//                  - int           mainKey         | main key selection        | 0=HKLM, 1=HKU, 2=HKCR
//                  - LPCTSTR       key             | key path string           | string
//                  - LPCTSTR       valueName       | value name                | string
//                  - LPDWORD       value           | pointer to return val     | pointer
// return value :   BOOL                            | TRUE if success, FALSE if error
BOOL readRegister(int mainKey, LPCTSTR key, LPCTSTR valueName, LPDWORD value)
{

        struct HKEY__ * mainKeyAddr;        // main key selection handle
        struct HKEY__ * keyAddr;            // current key handle
        int statusHandler;                  // register functions return codes
        BOOL success = FALSE;               // error detection
        DWORD valueLenght;                  // needed variable used to store read lenght
        DWORD valueType;                    // needed variable used to store read type
```

```
	// replace user mainKey choice by correct handle
	switch(mainKey)
	{
		case 0 :	mainKeyAddr = HKEY_LOCAL_MACHINE;	break;
		case 1 :	mainKeyAddr = HKEY_CURRENT_USER;	break;
		case 2 :	mainKeyAddr = HKEY_CLASSES_ROOT;	break;
		default :	mainKey=-1;	break;
	}

	// if valid main key selection
	if(mainKey!=-1)
	{

		// open selected register key and return handle to it (keyAddr)
		// in read mode (KEY_QUERY_VALUE)
		statusHandler = RegOpenKeyEx (mainKeyAddr,
						key, 0, KEY_QUERY_VALUE, &keyAddr);

		// if key found and opened
		if(statusHandler == ERROR_SUCCESS)
		{
			// read the last DWORD containing key informations
			do
			{
				statusHandler = RegQueryValueEx (keyAddr, valueName, 0, &valueType,
						(LPBYTE)value, &valueLenght);
			}
			while(statusHandler == ERROR_MORE_DATA);



			// control if modification is successed
			if(statusHandler == ERROR_SUCCESS)
			{
				success = TRUE;
			}

			// close register key
			RegCloseKey(keyAddr);
		}
	}

	return success;
}


// function which alert the WinCE kernel to restart system
// (without modification in file system or registry)
void kernel_restart(void)
{
	KernelIoControl(IOCTL_HAL_REBOOT, NULL, 0, NULL, 0, NULL);
}

// function used to replace all registry PXA display driver parameters
BOOL writeRegDisplayValues(int Bpp, int CxScreen, int CyScreen,
						int pclk, int hsw, int blw, int elw,
						int vsw, int bfw, int efw, int hsp,
						int vsp, int pcp, int abc)
{
	BOOL success = TRUE;		// error detection parameters

// display driver registry path
#ifdef MOD_PXA320
	LPCTSTR regPath = TEXT("Drivers\\Display\\Colibri");
#else
	LPCTSTR regPath = TEXT("Drivers\\Display\\PXA27x\\Config");
#endif

	if(!writeRegister(0, regPath, TEXT("Bpp"), Bpp))		success=FALSE;
	if(!writeRegister(0, regPath, TEXT("CxScreen"), CxScreen))	success=FALSE;
	if(!writeRegister(0, regPath, TEXT("CyScreen"), CyScreen))	success=FALSE;
	if(!writeRegister(0, regPath, TEXT("pclk"), pclk))		success=FALSE;
	if(!writeRegister(0, regPath, TEXT("hsw"), hsw))		success=FALSE;
```

```
        if(!writeRegister(0, regPath, TEXT("blw"), blw))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("elw"), elw))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("vsw"), vsw))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("bfw"), bfw))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("efw"), efw))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("hsp"), hsp))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("vsp"), vsp))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("pcp"), pcp))        success=FALSE;
        if(!writeRegister(0, regPath, TEXT("abc"), abc))        success=FALSE;
        return success;
}

BOOL setDisplayResolution(int resolution)
{
        BOOL success = TRUE;        // error detection parameters

        switch(resolution)
        {
                case 1:
                {
                        if(!writeRegDisplayValues(32, 800,600,34666666,64, 60,12, 1,14,
                                2,1,1,0,239))        // 800x600  - 60Hz
                                success=false;
                        break;
                }
                case 2  :
                {
                        if(!writeRegDisplayValues(32, 640,480,26000000,64, 25,22, 2,10,
                                10,1,1,0,255))        // 640x480  - 60Hz
                                success=false;
                        break;
                }
#ifdef MOD_PXA320
                case 3  :
                {
                        if(!writeRegDisplayValues(16,1024,768,69333333,64,114,44,36,20,
                                30,0,1,0,250))        // 1024x768 - 60Hz
                                success=false;
                        break;
                }
#else
                case 3  :
                {
                        if(!writeRegDisplayValues(24,1024,768,52000000,64, 96,48, 2,33,
                                10,0,1,0,252))        // 1024x768 - 50Hz
                                success=false;
                        break;
                }
#endif

                default : success=false;
        }

        return success;

}
```

Diagramme UML du programme principal



Toutes les classes sont des fenêtres indépendantes sauf « configFile » qui n'intègre aucune interface graphique.

Procédure pour l'initialisation et la lecture de données sur l'ISL6295 via I^2C

Introduction

Cette annexe a pour but d'expliquer précisément la méthodologie à suivre pour communiquer, configurer et récupérer des données de la puce de gestion batterie ISL6295 d'Intersil. Cette dernière est connectée sur le bus I^2C, qui la relie au module Colibri.

On partira du principe que les fonctions de la DLL RegAccess.dll qui permettent d'envoyer et de recevoir des données sur le bus sont fonctionnelles. Voici le détail de ces fonctions :

```
BOOL I2CWriteISL6295Reg(char slaveAddr, BOOL bank, char regAddr, char value)
```

Cette fonction de la DLL permet d'écrire un byte dans un des registres de l'ISL6295 via le bus I2C. Le premier paramètre contient l'adresse du destinataire, le second permet de choisir dans quelle zone de la mémoire du destinataire l'on veut écrire la donnée (EEPROM ou RAM), le troisième est l'adresse du registre dans lequel écrire et enfin le dernier la valeur sur 8 bits de cette données.

La valeur de retour de type booléen permet de savoir si la communication s'est bien déroulée. Il est possible d'avoir cette information car l'ISL6295 quittance les données tout au long de la procédure d'écriture. Ainsi, s'il ne le fait pas, la communication est arrêtée et la fonction renvoie une valeur « FALSE ».

```
int I2CReadISL6295Reg(char slaveAddr, int bank, char regAddr)
```

Cette fonction de la DLL permet de lire un byte contenu dans un des registres de l'ISL6295 via le bus I2C. Le premier paramètre contient l'adresse du destinataire, le second permet de choisir dans quelle zone de la mémoire du destinataire l'on veut lire la donnée (EEPROM ou RAM) et le troisième contient l'adresse que l'on veut lire.

La valeur de retour de type entier contient la donnée lue. Si une erreur s'est produite, elle renvoie la valeur -1.

L'ISL6295

Ce qu'il est important de savoir sur cette puce si l'on veut comprendre comment la programmer est l'organisation des registres. Ceux-ci sont atteignables par une adresse (sur 8 bits). Cette adresse correspondant à celle à mettre en paramètre dans les fonctions ci-dessus.

La mémoire de l'ISL6295 est divisée en trois parties. La première est une EEPROM, qui contient les données de configuration de base. Celles qui seront chargées à chaque mise sous tension de la puce. La deuxième est de la mémoire RAM, celle dans laquel la configuration d'origine est chargée. Elle contient donc la configuration active de la puce. Si l'on modifie des valeurs dans la RAM, on peut reconfigurer la puce directement.

On trouve aussi dans la RAM les registres de données qui contiennent les valeurs échantillonnées par le convertisseur A/D. Il faut savoir qu'il en existe 12. L'ISL6295 est en effet capable de mesurer de nombreux éléments comme la tension batterie, la température, le courant de charge et de décharge, la température d'une résistance externe, etc.

La troisième partie de la RAM contient des registres réservés dont l'accès depuis le bus I2C est prohibé.

L'initialisation de l'ISL6295

Lorsque l'appareil démarre, il est tout d'abord nécessaire d'initialiser la puce de gestion pour qu'elle soie dans un état connu. Il faut lui dire qu'elle doit commencer à échantillonner des données. Cela se fait en configurant un registre dans la mémoire RAM nommé « ADConfig » pour Analog/Ditial Configuration register. Son adresse est 0x43. Voici la description de ce registre :

A/D CONFIGURATION REGISTER – ADconfig
(Address - 43 Hex/67 Decimal)

7	6	5	4	3	2	1	0
ADEN	Samp	Reserved			SampDiv (2:0)		

- Le premier bit permet l'activation du convertisseur A/D. Il doit être mis à 1.
- Le 2ème bit permet de choisir si la puce doit travailler en mode Sample (échantillons pris chaque x secondes) ou en mode continu. On choisira le mode Sample (bit à 1), dans le but d'économiser de l'énergie.
- Les trois bits suivants sont réservés, leur valeur n'a pas d'importance.
- Les trois dernier bits configure la période d'échantillonnage. Avec une valeur de 1, on prélève un échantillon chaque seconde.

C'est donc cette configuration qui a été choisie pour l'iPresenter2. La valeur du registre en hexadécimal est de 0xC1.
Il faut savoir que l'adresse physique de la puce sur le bus I2C est de 0x13. Cette adresse est celle définie par défaut et il n'y a pas de raisons de la changer. C'est pour cela que dans le programme principal, à l'initialisation, on retrouve les lignes suivantes :

```
const byte I2C_SlaveAddr            = 0x13;
const byte I2C_SelBankRAM           = 1;
const byte I2C_RegAddr_ADConfig     = 0x43;
const byte I2C_RegVal_ADConfig      = 0xC1;

I2CWriteISL6295Reg((char)I2C_SlaveAddr, (char)I2C_SelBankRAM,
    (char)I2C_RegAddr_ADConfig, (char)I2C_RegVal_ADConfig);
```

On va donc écrire la valeur 0xC1 dans le registre 0x43 de la RAM.

La 2ème opération consiste à configurer le convertisseur A/D pour qu'il échantillonne la tension de la batterie. Cela se fait dans le registre nommé « VPCtrl » pour Voltage Pack Control Register situé à l'adresse 0x4E. Voici la description de ce registre :

7	6	5	4	3	2	1	0
Enable	Resolution			Reference	Select		

Le premier bit permet d'activer l'échantillonage pour la tension batterie. Il doit être mis à 1.
Les trois bits suivants permettent de choisir la résolution dans laquelle doit être prise la mesure. Une mesure sur 10 bit est suffisante, soit la valeur 2.
Le bit 3 permet de choisir la référence de tension à utiliser. Elle doit être de 340mV (CF. Datasheet ISL6295 page 7, chapitre « Voltage Measurements ». Valeur du registre à utiliser : 1
Les trois derniers bits permettent de choisir la source de la mesure, soit dans notre cas la tension de la batterie. La valeur correspondante est 3.

C'est donc cette configuration qui a été choisie pour l'iPresenter2. La valeur du registre en hexadécimal est de 0xAB.

C'est pour cela que dans le programme principal, à l'initialisation, on retrouve les lignes suivantes :

```
const byte I2C_SlaveAddr            = 0x13;
const byte I2C_SelBankRAM           = 1;
const byte I2C_RegAddr_VPackConfig  = 0x4E;
const byte I2C_RegVal_VPackConfig   = 0xAB;

    I2CWriteISL6295Reg((char)I2C_SlaveAddr, (char)I2C_SelBankRAM,
        (char)I2C_RegAddr_VPackConfig, (char)I2C_RegVal_VPackConfig);
```

On va donc écrire la valeur 0xAB dans le registre 0x4E de la RAM.

Lecture d'une valeur depuis les registres de données

Maintenant que la puce est configurée correctement, elle va automatiquement échantillonner des données chaque seconde dans les registres de données. Ceux-ci sont pour la tension batterie à l'adresse 0x4C (LSB) et 0x4D (MSB). La valeur sera sur 10 bits.

C'est pour cela que dans le programme principal, à l'initialisation, on retrouve les lignes suivantes :

```
        const byte I2C_SlaveAddr            = 0x13;
        const byte I2C_SelBankEEPROM        = 0;
        const byte I2C_SelBankRAM           = 1;
        const byte I2C_RegAddr_VPackMSB     = 0x4D;
        const byte I2C_RegAddr_VPackLSB     = 0x4C;

// get 16 bit from 2x8 bits of battery voltage value registers

voltageRegister = (I2CReadISL6295Reg((char)I2C_SlaveAddr, (char)I2C_SelBankRAM,
(char)I2C_RegAddr_VPackMSB) << 8); // shifted of 8 bits (MSB)

voltageRegister += (I2CReadISL6295Reg((char)I2C_SlaveAddr, (char)I2C_SelBankRAM,
(char)I2C_RegAddr_VPackLSB));   // LSB, simply added
```

Le programme va tout d'abord chercher le contenu des 8 bits du MSB et le stocker dans une variable locale de type entier (32 bits) après les avoir multiplié par 8. Il va ensuite chercher le LSB et l'ajouter à la variable précédemment utilisée. On obtient donc une valeur sur 10 bits qui représente la tension de la batterie.

Remarque

Il n'est pas nécessaire de réinitialiser les registres de configuration à chaque démarrage de l'iPresenter2. Cela est fait uniquement pour être certain qu'elle soit dans un état connu. En effet, la tension batterie alimentant toujours la puce même quand l'appareil est éteint, sa configuration ne devrait jamais être perdue.

Cette sécurité permet d'éviter que la puce ne réponde plus si la batterie est absolument vide après une période d'inutilisation prolongée.