

Banc de test Arduino des objectifs Canon EF-Eos



(Pour éviter les prises de têt)

Table des matières

<i>Présentation</i>	<i>1</i>
<i>Avant propos</i>	<i>1</i>
<i>Principe de fonctionnement</i>	<i>1</i>
<i>Notice d'utilisation</i>	<i>1</i>
1 : Insertion	2
2 : Caractéristiques	2
3 : Nom objectif	2
4 : Caractéristiques d'ouverture de l'objectif	2
5 : Focale en cours	2
6 : Distance réelle de mise au point	2
7 a 12 : Affichage des registres de fonctionnement de l'objectif.	2
13 : Test du diaphragme	3
14 : Commande du diaphragme en pas à pas	3
15 : Test moteur autofocus	3
16 et 17 : Commande du moteur autofocus en pas a pas.	3
18 : Commande stabilisation	3
19 : Réglage vitesse moteur AF	3
20 : Eclairage EF	3
21 : Trace SPI-EF	3
<i>Version I</i>	<i>4</i>
<i>Interface Physique de test</i>	<i>4</i>
Alimentation	5
Interface SPI	5
<i>Software</i>	<i>7</i>
<i>Structure programme</i>	<i>7</i>
Horloges et durées de fonctionnement	7
Gestion des touches	7
Gestion des menus de test	8
Transmission des données a l'objectif	8
Contrôle d'envoi des données	8
Principales fonctions auxiliaires diverses	8
Synoptique de fonctionnement d'une séquence de test	9
<i>Listing complet</i>	<i>10</i>
Boucle principale	10

Cmd_EF	13
Test_Auto.ino	14
Test_KeyH.ino	16
Test_Select.ino	18
Test_Solo.ino	19
SPIxEos.h	20
SPIxEos.ino	21
<i>Version II</i>	<i>23</i>
<i>Interface Physique de test</i>	<i>23</i>
Schéma interface	24
Connecteurs interface	24
<i>Réalisation pratique</i>	<i>25</i>
Interface Arduino	25
Adaptateur EF	25
<i>Software</i>	<i>26</i>
<i>Structure programme</i>	<i>26</i>
<i>Listing complet</i>	<i>27</i>
Boucle principale	27
Cmd_EF	30
Test_Auto.ino	31
Test_KeyH.ino	33
Test_Select.ino	35
Test_Solo.ino	36
SPIxEos.h	38
SPIxEos.ino	38
<i>Liens et révisions document</i>	<i>40</i>

Présentation

Avant propos

De nombreux problèmes de fonctionnement d'objectifs sont imputables à un défaut interne. Si la classique Erreur01 causée par une rupture de la nappe de liaison électrique du diaphragme est facile à détecter, des problèmes de performance d'autofocus pouvant être dus à la panne d'un codeur ou d'un détecteur interne sont nettement moins faciles à diagnostiquer.

Le protocole informatique Eos-EF de dialogue entre les boîtiers et les objectifs interchangeables de la marque Canon permet de disposer d'un grand nombre d'informations en provenance de l'objectif ou permet d'en commander séparément chaque actuateur. Il sera donc possible de tester indépendamment de tout contexte externe les différents éléments d'un objectif, tout en contrôlant leur fonctionnement grâce aux informations de retour qui leur sont liés.

Une platine Arduino accompagnée d'une interface électronique relativement simple permet de réaliser ce banc de test. Pour plus de détails sur le protocole EF et ses caractéristiques électriques la lecture du document dédié sera conseillée.

Principe de fonctionnement

Les deux versions décrites dans ce document sont quasiment semblables dans leur principe de fonctionnement, seul l'interfaçage physique avec l'objectif fera l'objet de différences sensibles. La version 1 utilisera l'adaptateur utilisé pour l'analyse et le décodage du protocole EF, la version 2 disposera d'une interface plus adaptée à un usage en stand alone.

Dans les deux cas, le cœur du système sera basé sur une platine Arduino Mega dotée d'un shield LCD - 5 boutons standard.

Notice d'utilisation

- L'afficheur LCD est divisé en deux zones, la ligne supérieure affiche un descriptif succinct du test en cours, la ligne inférieure son résultat ou les informations s'y reportant.
- Les touches verticales permettent le passage au test précédent ou suivant.
- Les touches horizontales ou Select en fonction du contexte du test lanceront l'action désirée.

L'afficheur étant limité en taille, une sortie des informations pourra être émise de façon plus détaillée sur le port série à la vitesse de 115200 bauds.

Deux entrées optionnelles de la carte Arduino permettent pour l'une d'envoyer sur le port série le contenu du buffer d'échange EF entre l'objectif et le banc de test, pour l'autre d'effectuer un arrêt rapide des tensions d'alimentation de cette optique.

1 : Insertion

Affiche l'état de l'alimentation générale de l'objectif. La touche *Select* permet soit de couper toute source de tension dans le but de son retrait, soit si celles-ci sont déjà inhibées de lancer un cycle d'initialisation et de détection de la présence effective d'un objectif.

Toute anomalie ou erreur de transmission lors des tests provoqueront l'extinction des tensions d'alimentation de l'objectif et le retour a ce menu.

2 : Caractéristiques

Affiche sous forme brute au format hexadécimal les 8 octets décrivant les caractéristiques et fonctionnalités principales de l'objectif. Ces valeurs sont décodées et envoyées sur le port serie tout comme les 4 octets de paramétrage optionnels si l'objectif en dispose.

3 : Nom objectif

Affiche les 16 premiers caractères sur le LCD et envoie sur le port serie la valeur complète du nom usuel de l'objectif. Cette fonctionnalité n'étant apparue que sur des objectifs relativement récents elle est conditionnée aux résultats du test précédent.

4 : Caractéristiques d'ouverture de l'objectif

Affiche les valeurs d'ouverture minimale et maximale offertes par l'objectif. Cet affichage est mis à jour automatiquement pour les objectifs dotés d'une ouverture glissante.

5 : Focale en cours

Affiche de façon continue la focale de l'objectif en mm, cette valeur ne peut changer évidemment que sur un objectif de type zoom. Ceci permet de tester rapidement tout défaut dans le fonctionnement du codeur de position de la bague de changement de focale.

6 : Distance réelle de mise au point

Affiche pour les objectifs disposant de la fonction C2 les valeurs approchées (plus et moins) de la distance réelle de mise au point en centimètres.

7 a 12 : Affichage des registres de fonctionnement de l'objectif.

Affiche sous forme hexadécimale les valeurs internes de fonctionnement de l'objectif.

- B2 : 3 octets de paramétrage de l'AF dépendant de l'objectif et de la focale en cours.
- E0 : 2 octets issus de l'encodeur de position du zoom.
- E4 : 2 octets (Lens extension factor ?)
- C4 : 2 octets issus de l'encodeur de distance de mise au point.
- E8 : 6 octets
- EA : 6 octets

13 : Test du diaphragme

Une action sur les touches horizontales permet de passer de la position pleine ouverture à celle iris totalement fermé ou inversement. Le temps de fonctionnement du moteur est affiché permettant de diagnostiquer un éventuel problème si des différences apparaissent entre ces deux opérations. Le fonctionnement du détecteur de position PO est également contrôlé, soit sa disparition lors de la fermeture de l'iris, soit sa présence lors du retour à pleine ouverture.

14 : Commande du diaphragme en pas à pas

Les touches horizontales permettent de manœuvrer l'iris par pas de $1/8^e$ d'IL, le compteur affiché est une valeur logique calculée en fonction du nombre d'impulsion envoyées, seule la position PO est assurée physiquement. La sélection initiale de ce menu provoque automatiquement le passage en position PO du diaphragme.

15 : Test moteur autofocus

Les touches horizontales provoquent le déplacement du moteur AF de la position infini à la distance de map mini ou inversement. Le temps réel de fonctionnement du moteur est affiché comme pour le diaphragme.

16 et 17 : Commande du moteur autofocus en pas à pas.

Ces tests permettent de commander un déplacement de 16 pas du moteur AF dans un sens ou dans l'autre à l'aide des touches horizontales.

Ils se différencient par l'affichage de la valeur de positionnement des lentilles de mise au point, le test C2 affiche la distance de MAP réelle, le test C0 affiche la valeur du compteur relatif interne, la touche *Select* permettant de le remettre à zéro.

Dans les deux cas l'affichage d'une barre oblique devant une de ces valeurs signifie que le moteur AF est en butée sur la position MAP mini ou infini. Certains objectifs dans cette configuration ne disposent pas de coupure du moteur arrivé en butée, des envois d'ordres de déplacement incorrects peuvent endommager l'objectif.

18 : Commande stabilisation

La touche *Select* permet de lancer un cycle d'activation pendant une seconde de la stabilisation de l'objectif si celui-ci le permet. L'état du module IS est affiché sur la seconde ligne du LCD.

19 : Réglage vitesse moteur AF

Pour les objectifs disposant d'une vitesse de moteur autofocus variable, les touches horizontales permettent de choisir la vitesse utilisée dans les tests.

20 : Eclairage EF

Permet de mettre En ou Hors fonction la Led d'éclairage interne de l'objectif située sur l'interface de test.

21 : Trace SPI-EF

Activée cette option permet d'envoyer sur le port série la totalité du dialogue SPI échangé entre l'objectif et le banc de test sous la forme Octet Envoyé - Reçu - Temps de validation commande par objectif.

Version I



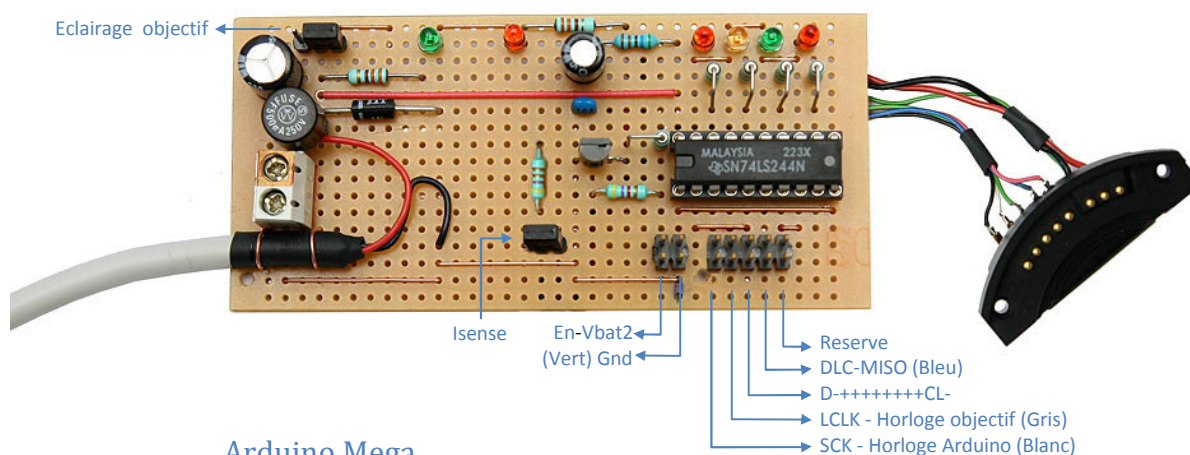
Interface Physique de test

Le principal risque lors de la manipulation des boîtiers ou des objectifs sera celui de créer un court-circuit et de provoquer la destruction d'un de ces deux éléments. Il sera donc préférable d'utiliser des connecteurs EF standards non DIY, soit en modifiant une bague allonge à bas coût, soit comme ici en utilisant la carcasse d'un vieux boîtier argentique de récupération.

Dans cet exemple un large morceau de circuit imprimé d'expérimentation est fixé sur le plan film de ce boîtier démunie de l'ensemble de ses composants internes, laissant une place généreuse pour les composants.



Implantation et connexion a la CPU Arduino



Arduino Mega

		+5v	+5v		
(Bleu) DLC	PA0	22	23	PA1	LCLK (Gris)
	PA2	24	25	PA3	
(réserve) En-Vbatt2	PA4	26	27	PA5	En-Vbatt2 (Jaune)
(Blanc) SCK	PA6	28	29	PA7	DCL (Violet)
(Non utilisé) En_Vcc		52	53		Led Diag +
(Vert) Gnd		0v	0v		Led Diag -

Seul le connecteur d'extension digital de la carte Mega est utilisé. La liaison avec le module de test est réalisée avec des cordons HE10 male-femelle.

Une Led avec une résistance série de 330Ω pourra être câblée entre la sortie

Dig53 et la masse, celle-ci permet de contrôler le fonctionnement de la carte CPU en clignotant au rythme de 1.5Hz.

Structure programme

Pour simplifier la gestion du code celui-ci est reparti sur plusieurs onglets :

- [*Canon_Test.ino*](#) : Déclaration tables et variables, boucle principale, gestion des menus.
- [*Fonctions.ino*](#) : Divers sous-programmes communs, gestion du dialogue EF.
- [*Cmd_EF.ino*](#) : Commandes envoyées à l'objectif utilisées plusieurs fois.
- [*Test_Solo.ino*](#) : Operations effectuées a chaque changement de menu.
- [*Test_Auto.ino*](#) : Operations effectuées régulièrement automatiquement.
- [*Test_KeyH.ino*](#) : Operations effectuées à chaque appui sur une touche droite ou gauche.
- [*Test_Select.ino*](#) : Operations effectuées a chaque appui sur la touche select.
- [*SPIxEos.h + .ino*](#) : Bibliothèque de dialogue série au format EF (Voir le document dédié).

Horloges et durées de fonctionnement

Les événements, temps de fonctionnement ou temporisations du programmes utilisent directement l'horloge interne [*millis\(\)*](#) de la carte Cpu. A chaque événement cyclique est associé un compteur mis a l'heure courante lors de son exécution, cette mémorisation est comparée en permanence dans la boucle principale aux nouvelles valeurs d'heure courante, si la différence entre les deux excède une certaine valeur l'événement est a nouveau déclenché. Cette méthode est utilisée pour les événements suivants :

- Intervalle entre scan clavier (25ms)
- Intervalle entre opérations de test automatique ([*RepeatAuto*](#)=200ms).
- Durée d'activation minimale de la stabilisation ([*TempoStab*](#)=1s) a laquelle il faut rajouter le délai avant extinction de l'objectif.
- Durée du maintient actif de la tension de puissance objectif en l'absence de tout dialogue ([*TempoAlimMot*](#)=2s)
- Durée avant passage a l'état repos/inactif de l'objectif en l'absence de dialogue ou d'action sur une touche ([*TempoAlimEF*](#)=3mn).
- Fréquence de clignotement de la Led chien de garde (0.75s).

Gestion des touches

Le shield LCD standard utilise une gestion des touches analogiques, à chaque action sur une touche correspond un niveau de tension sur l'entrée analogique 0 de la carte Cpu. Le décodage est réalisé par comparaison avec les constantes [*Keyxxx*](#).

Les événements liés à chaque touche ne sont réalisés qu'une seule fois par mémorisation de la valeur de la touche active dans la variable [*MemoKey*](#). Une répétition automatique des touches est effectuée grâce a un compteur incrémenté tant que la touche est active ([*CptRepeatKey*](#)), une comparaison avec deux constantes simule a échéance un relâchement de la touche. La constante [*RepeatKeyL*](#) gère le temps avant activation de la répétition, la constante [*RepeatKeyH*](#) l'intervalle de temps entre chaque répétition une fois celle-ci débutée.

Le choix du menu de test en cours est géré par la variable IdxMenu incrémentée ou décrémentée par une action sur les touches *Hauts* ou *Bas*. Pour éviter de renommer l'ensemble des menus en cas d'insertion d'un nouvel élément dans la liste, la table contenant le nom des menus à afficher NomMenu[] et les sous programmes de test passent par la table OrdreMenu[IdxMenu] établissant la correspondance entre l'ordre numérique donné par IdxMenu et l'ordre logique d'exécution ou d'affichage sur le LCD.

A chaque menu de test et touche correspond une séquence de programme, seule les séquences à exécution automatique sont conditionnées à la présence d'un dialogue correct entre la carte CPU et l'objectif par l'intermédiaire du flag EF_InitOk.

Transmission des données à l'objectif

La fonction SendBuff_EF permet d'envoyer une suite de commandes à l'objectif par l'intermédiaire de la bibliothèque SpixEos. Pour ce, un buffer double émission et réception de 32 caractères maximum est utilisé. Du fait de l'utilisation d'une majorité de zéro dans la syntaxe des commandes du protocole EF (en particulier lors d'une demande de paramètres) plutôt que d'envoyer une chaîne de commande complète seuls les octets différents de zéro sont injectés dans le buffer avant émission. La longueur de la trame à émettre sera passée en paramètre lors de l'appel de la fonction.

Contrôle d'envoi des données

Contrairement à un boîtier Eos l'adaptateur n'ayant pas de détection physique du retrait ou de l'insertion d'un objectif et pour éviter les contrôles redondants, deux flags sont utilisés, EF_AlimOk mis à 1 tout le temps de présence de la tension d'alimentation de l'informatique de l'objectif Vcc, et EF_InitOk mis à 1 si la séquence d'initialisation et de lecture des caractéristiques principales de l'objectif se sont correctement déroulées.

La procédure d'arrêt Arret_EF activée soit par écoulement du temps de veille de l'objectif, soit en cas d'échec de l'initialisation remet ces deux flags à zéro et coupe toute source d'alimentation.

Avant chaque envoi d'une trame de commande à l'objectif le flag EF_InitOk est examiné, s'il est à zéro une séquence d'initialisation est effectuée. Chaque trame débute par l'envoi d'une commande de demande de disponibilité de l'objectif 0x0A-00, si la réponse diffère de 0xAA-00 la séquence de test est annulée et le flag EF_InitOk mis à zéro pour lancer une séquence d'initialisation à la prochaine demande de dialogue avec l'objectif.

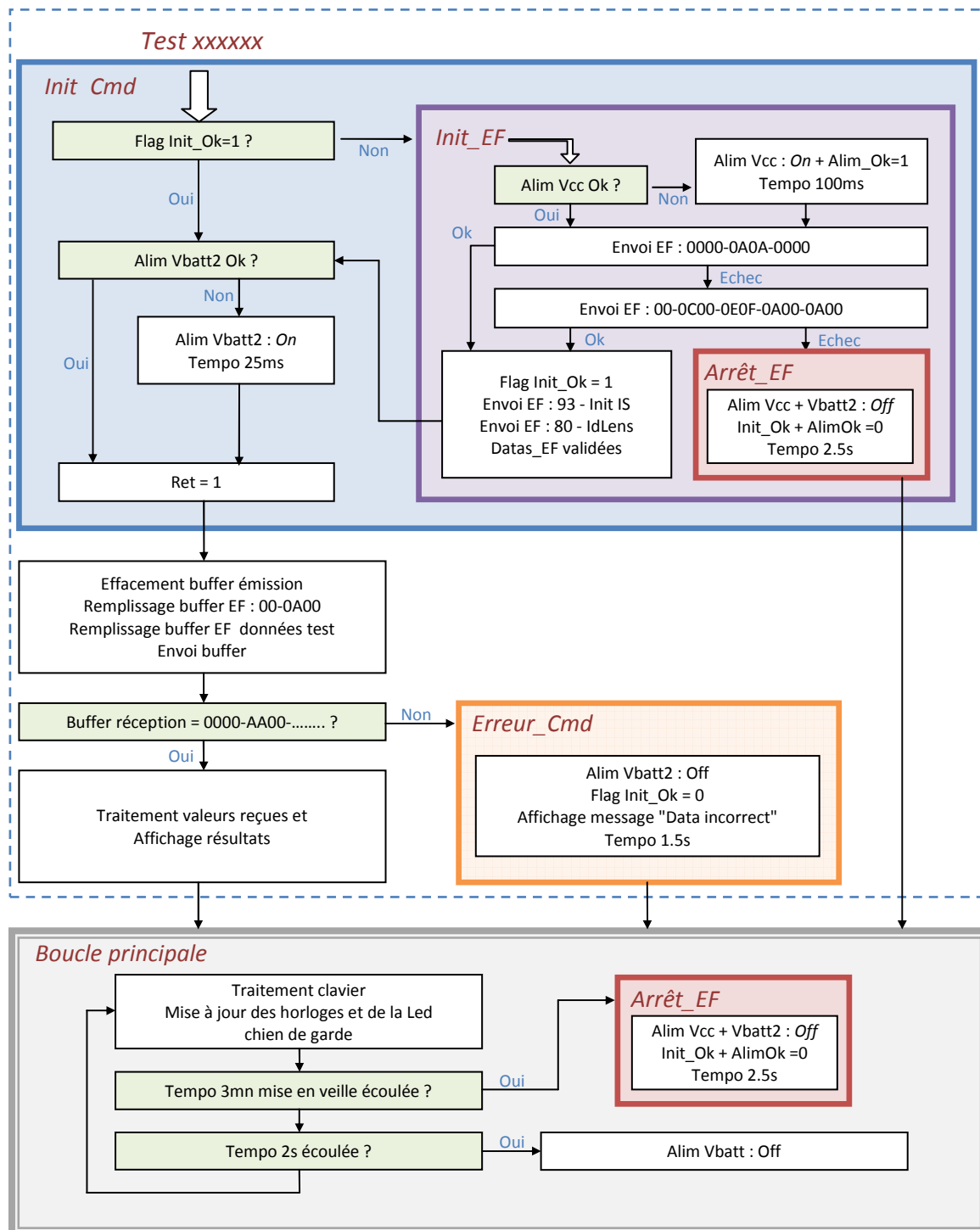
La séquence d'initialisation Init_EF est composée d'un contrôle de la présence de la tension d'alimentation de l'optique Vcc, suivi de l'envoi d'une commande de réveil standard 0x0A, si celle-ci échoue un reset complet de l'objectif est tenté, un nouvel échec provoque un arrêt complet de l'objectif, suivie d'une temporisation de blocage de 2.5s.

Principales fonctions auxiliaires diverses

- Init_EF : Initialisation de l'objectif et mémorisation de ses caractéristiques principales.
- Init_Cmd : Vérification de l'alimentation et l'initialisation de l'objectif (Retour 1=Ok).
- SendBuff_EF : Envoi le contenu du buffer l'objectif.
- RazBuff_EF : Effacement du buffer émission.
- RazData_EF : Effacement des caractéristiques de l'objectif mémorisées.

- Arrêt_EF : Mise a l'arrêt de l'objectif et de ses tensions d'alimentation.
- Erreur_Cmd : Gestion en cas de réception d'une valeur inattendue après la transmission d'une trame de commande
- RefreshMenu : Efface l'afficheur LCD et imprime le nom du menu en cours.
- LceHex8 : Affiche sur le LCD un octet au format hexa sur 2 caractères.
- PrintHex8 : Envoie sur le port série un octet au format hexa 2 caractères plus séparateur.
- PrintHexBin8 : " " Hexa 2 caractères plus binaire.
- Debug_A : Envoie au port série le contenu des buffers et l'état des flags système.

Synoptique de fonctionnement d'une séquence de test



Listing complet

Le listing qui suit n'est donné qu'à titre indicatif dans le cas d'un Pdf orphelin, seule la dernière version disponible ici sera à prendre en compte.

[http://jp79dsfr.free.fr/ Docs%20et%20infos/Sources/Protocole%20EF/Sketchs/Canon_Test_objectif_v1.zip](http://jp79dsfr.free.fr/Docs%20et%20infos/Sources/Protocole%20EF/Sketchs/Canon_Test_objectif_v1.zip)

Boucle principale

```
// =====
//=
//=      Logiciel de diagnostic et test du fonctionnement des objectifs EOS Canon
//=
//= Chargement sur platine Mega uniquement avec shield LCD-Boutons standard
//=                                     © JP 2016 =
// =====

//===== Déclarations
#include <LiquidCrystal.h>
#include "SPIxEOS.h"

LiquidCrystal Lcd(8,9,4,5,6,7); //----- Définition paramètres shield clavier - LCD
const char LcdNull="";
const uint8_t PortAnKey = 0;
uint16_t AnalogKey = 0;
uint8_t ValKey = 0;
uint8_t MemoKey = 0x20;
const uint8_t KeyHaut =0x5;
const uint8_t KeyBas =0xA;
const uint8_t KeyDroit =0x1;
const uint8_t KeyGauche =0xF;
const uint8_t KeySel =0x17;
const uint8_t KeyNone =0x20;

const uint8_t LedCDG = 53; //----- Définitions ports E/S
const uint8_t LedTest= 50;
const uint8_t LedLens= 51;
const uint8_t VccEF= 52;
const uint8_t Bp_ATU=20;
const uint8_t Bp_Debug=21;
//Interface EF sur PortA, modification logiciel si changement

uint8_t CliLedCDG=0; //----- Variables temporisations cycle et chien de garde
uint32_t CptLedCDG;
uint8_t CliLedTest=0;

const uint16_t RepeatAuto=200; //Intervalle entre test Auto en ms
uint32_t CptTempoAuto;

const uint8_t RepeatKeyL=25; //Intervalle avant début répétition touche x 25ms
const uint8_t RepeatKeyH=2; //Intervalle entre répétitions touche x 25ms
uint32_t CptTempoKey;
uint8_t CptRepeatKey=1;
uint8_t RepeatKey=RepeatKeyL;

const uint32_t TempoAlimEF= 180000; //Durée maintient alim objectif apres dernier envoi
uint32_t CptAlimEF;
const uint16_t TempoAlimMot= 2000; //Durée maintient Vbatt2 apres dernier envoi
uint32_t CptAlimMot;
const uint16_t TempoStab= 1000; //Durée minimale action ls
uint32_t CptStab;
uint32_t ValTemps;

//----- Gestion et définitions des menus
const uint8_t MaxMenu=20; //Nb de menu max -1
uint8_t IdxMenu=0; //Index Menu en cours (de 0 a MaxMenu)
const uint8_t OrdreMenu[21]={0,1,2,3,4,6,7,5,8,19,17,18,9,10,11,12,13,20,14,15,16}; //Ordre d'affichage
menus
const char* NomMenu[21]={
    " Insertion",
    "Caractéristiques",
    " Nom objectif",
    " Av Min/Max",
    "Focale actuelle",
    "Codeur Zoom E0",
    "Distance map C2",
    "Best AF adj. B2",
```

```
"Lens ExtFact. E4",
" Cmd AV MAX/min",
" Cmd AV continu",
"Cmd AF mini/inf.",
"Cmd. AF Lin - C0",
"Cmd. AF Lin - C2",
"Adj. Vitesse AF",
" Eclairage EF",
" Trace SPI-EF",
"Registre AF E8",
"Registre AF EA",
"Codeur AF C4",
" Commande IS"
""};
```

```
const char* MessLcd[28] ={
"Banc test EOS-EF",
"v1.00",
" Tension Off ",
" Tension ON ",
"Init objectif Ok",
" Pas d'objectif ",
"Serial debug Off",
"Serial debug On",
" Led EF : Off ",
" Led EF : On ",
" Vitesse AF=$ ",
"Erreur réception",
" Cmd inconnue ",
" Non disponible ",
" AF en manuel ",
" AF Ok : ",
" Erreur AF ",
" Valeur iris: ",
" Tps fct: ",
"Erreur switch PO",
"Objectif sans IS",
" Inter IS Off ",
" IS a l'arret ",
" IS en fonction ",
" Initialisation ",
"(de)Verrouillage",
" Statut= ",
""};
```

```
// ----- Valeurs Dialogue EF
uint8_t EF_InitOk=0; //Flag optique présente
uint8_t EF_AlimOk=0; //Flag présence alim Vcc 5v objectif
```

```
uint8_t EF_TypeLens; //Memo caractéristiques Lens
uint8_t EF_IdLens;
uint16_t EF_Fmin;
uint16_t EF_Fmax;
uint8_t EF_Fct1;
uint8_t EF_Fct2;
uint8_t EF_AvT;
uint8_t EF_AvMin;
uint8_t EF_AvMax;
uint8_t EF_CptAv;
```

```
uint8_t IdxVitesseAF=3; //Tables valeurs vitesse moteur AF
uint8_t VitesseAF[6]={0,0x2C, 0x2D, 0x2E, 0x2F, 0xFF};
```

```
uint8_t BuffSendEF[32]; //Buffer Emission réception
SPIEOS_Ret BuffRecEF[32];
```

```
uint8_t FlagDebug=0; // Pour test et debug comm EF
uint8_t ValChange8; // Pour memo variation de valeurs
uint16_t ValChange16;
uint8_t Tmp8a; // Variables temporaires et de travail
uint8_t Tmp8b;
uint16_t Tmp16a;
uint8_t Cpt;
uint16_t Cpt16;
```

```
// ===== Initialisation
```

```
void setup() {
Lcd.begin(16,2);
Lcd.clear();
Lcd.setCursor(0,0);
Lcd.print(MessLcd[0]);
Lcd.setCursor(5,1);
Lcd.print(MessLcd[1]);
```

```
DDRA=0xF0; //Init entrées sorties
PORTA=0xCF;
pinMode(VccEF, OUTPUT);
digitalWrite(VccEF, LOW);
pinMode(LedCDG, OUTPUT);
digitalWrite(LedCDG, LOW);
pinMode(LedTest, OUTPUT);
digitalWrite(LedTest, LOW);
pinMode(LedLens, OUTPUT);
digitalWrite(LedLens, LOW);
pinMode(Bp_ATU, INPUT);
digitalWrite(Bp_ATU, HIGH);
pinMode(Bp_Debug, INPUT);
```

```

digitalWrite(Bp_Debug, HIGH);

Serial.begin(115200);
Serial.print("\r\nLogiciel de Test du fonctionnement des objectifs EOS Canon _v1.00\r\n\n");

delay (2500);
CptAlimEF=0xFFFF;
CptAlimMot=0xFFFF;
CptStab=0xFFFF;
RefreshMenu();
RazData_EF();
}

// ===== Boucle principale
void loop() {
  ValTemps=millis();

  if (ValTemps-CptLedCDG > 750){ // ----- Chien de garde
    CliLedCDG = !CliLedCDG;
    digitalWrite(LedCDG, CliLedCDG);
    CptLedCDG=ValTemps;
  }

  if (ValTemps - CptAlimMot > TempoAlimMot) { // ----- Coupure auto Vbatt2 apres envoi
    PORTA=0xCF;
  }

  if (EF_AlimOk==1) {
    if (ValTemps - CptAlimEF > TempoAlimEF) { // ----- Extinction auto interface EF apres touche
      Arret_EF();
    }
  }

  if (ValTemps-CptTempoKey > 25){ // ----- Gestion des Touches
    CptTempoKey=ValTemps;
    AnalogKey = analogRead(PortAnKey);
    ValKey = (AnalogKey / 32) + 1;

    if (MemoKey != ValKey){ //Changement de touche ou répétition
      MemoKey = ValKey;
      CptAlimEF=ValTemps;

      if (ValKey == KeyHaut) {
        if (IdxMenu==0) {IdxMenu=MaxMenu;}
        else {IdxMenu--;}
        RefreshMenu();
      }

      if (ValKey == KeyBas) {
        if (IdxMenu==MaxMenu) {IdxMenu=0;}

```

```

        else {IdxMenu++;}
        RefreshMenu();
      }
    }

    if (ValKey == KeySel) {Test_Select();}

    if (ValKey == KeyGauche) {Test_KeyH(0);}
    if (ValKey == KeyDroit) {Test_KeyH(1);}
  }

  else { // ----- Répétition touches
    if (ValKey == KeyNone) {
      RepeatKey = RepeatKeyL;
      CptRepeatKey=1; }
    else {
      CptRepeatKey ++;
      if (CptRepeatKey == RepeatKey) {
        MemoKey = KeyNone;
        CptRepeatKey=1;
        RepeatKey = RepeatKeyH; }
    }

    Tmp8a=digitalRead(Bp_ATU); // ----- Bouton arret d'urgence
    if (Tmp8a==LOW) {Arret_EF();}
    Tmp8a=digitalRead(Bp_Debug); // ----- Bouton debug
    if (Tmp8a==LOW) {Debug_A();}
  }

  if (EF_InitOk!=0){ // ----- Tests automatiques
    if (ValTemps-CptTempoAuto > RepeatAuto){
      CptTempoAuto=ValTemps;
      Test_Auto();
    }
  }
}

```

```
// =====
// = Commandes envoyées a l'objectif
// =====

void Cmd_13PO() { // ----- Ouverture Iris a PO
    Cmd_B0();
    if (EF_AvT!=0) {
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x13;
        BuffSendEF[4]=0x80;
        SendBuff_EF(8);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            Lcd.print(MessLcd[17]);
            Lcd.setCursor(13,1);
            if ((Cmd_90() & 0x01) ==0) {
                EF_CptAv=EF_AvMax;
                Lcd.print("PO");
            }
        }
        else { ErreurCmd(); }
    }
}

void Cmd_80(){ // ----- Récupération paramètres objectif
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0x80;
    BuffSendEF[4]=0x0A;
    BuffSendEF[5]=0x99; //Emulation 40D
    BuffSendEF[6]=0x01;
    SendBuff_EF(17);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        EF_TypeLens=BuffRecEF[4].ValRet;
        EF_IdLens=BuffRecEF[5].ValRet;
        EF_Fmin=(BuffRecEF[6].ValRet * 0x100) + BuffRecEF[7].ValRet;
        EF_Fmax=(BuffRecEF[8].ValRet * 0x100) + BuffRecEF[9].ValRet;
        EF_Fct1=BuffRecEF[10].ValRet;
        EF_Fct2=BuffRecEF[11].ValRet;
    }
    else {
        RazData_EF();
        ErreurCmd();
    }
}

}

uint8_t Cmd_90() { // ----- Demande statut AF/AV
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[2]=0x90;
    SendBuff_EF(6);
    return BuffRecEF[4].ValRet;
}

void Cmd_B0() { // ----- Récupération caractéristiques Av objectif

    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0xB0;
    SendBuff_EF(8);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        EF_AvT=BuffRecEF[4].ValRet;
        EF_AvMax=BuffRecEF[5].ValRet;
        EF_AvMin=BuffRecEF[6].ValRet;
    }
    else {
        EF_AvT=0;
        EF_CptAv=0;
        ErreurCmd(); }
}

void Cmd_Aff2x(uint8_t Val) { // ----- Affiche les 2octets retournés par Val
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=Val;
    SendBuff_EF(7);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        Lcd.print("Oct1:");
        LcdHex8(BuffRecEF[4].ValRet);
        Lcd.setCursor(9,1);
        Lcd.print("Oct2:");
        LcdHex8(BuffRecEF[5].ValRet);
    }
    else { ErreurCmd(); }
}
}
```

```
void Cmd_Aff6x(uint8_t Val) { // ----- Affiche les 6octets retournés par Val
  RazBuff_EF();
  BuffSendEF[1]=0x0A;
  BuffSendEF[3]=Val;
  SendBuff_EF(11);
  if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
    Lcd.print(" ");
    LcdHex8(BuffRecEF[4].ValRet);
    LcdHex8(BuffRecEF[5].ValRet);
    Lcd.print("-");
  }
```

```
    LcdHex8(BuffRecEF[6].ValRet);
    LcdHex8(BuffRecEF[7].ValRet);
    Lcd.print("-");
    LcdHex8(BuffRecEF[8].ValRet);
    LcdHex8(BuffRecEF[9].ValRet);
  }
  else { ErreurCmd(); }
}
```

Test_Auto.ino

```
// =====
// = Tests exécutés a intervalles réguliers automatiquement
// =====
```

```
void Test_Auto() {

  Lcd.setCursor(0,1);

  switch (OrdreMenu[IdxMenu]) {

    case 3: { // ----- Récupération AV min/max (B0)
      if (InitCmd()==0) {break;}
      Cmd_B0();
      if (EF_AvT!=0) {
        Lcd.print(" Max:");
        Lcd.print(EF_AvMax);
        Lcd.print(" Min:");
        Lcd.print(EF_AvMin);
      }
    }
    break;

    case 4: { // ----- Récupération focale actuelle (A0)
      if (InitCmd()==0) {break;}
      RazBuff_EF();
      BuffSendEF[1]=0x0A;
      BuffSendEF[3]=0xA0;
      SendBuff_EF(7);
      if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
```

```
        Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
        Lcd.setCursor(5,1);
        Lcd.print(Tmp16a);
        Lcd.print(" mm ");
      }
      else { ErreurCmd(); }
    }
    break;
```

```
    case 5: { // ----- Récupération valeur codeur zoom (E0)
      if (InitCmd()==0) {break;}
      Cmd_Aff2x(0xE0);
    }
    break;
```

```
    case 6: { // ----- Récupération distance map réelle (C2)

      if ((EF_Fct1 & 0x01) == 0x01) {
        if (InitCmd()==0) {break;}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0xC2;
        SendBuff_EF(9);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
          if (BuffRecEF[4].ValRet==0xC2) {Lcd.print(MessLcd[12]);}
          else {
            Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
            AffDmap(Tmp16a);
            Lcd.setCursor(8,1);
            Tmp16a=(BuffRecEF[6].ValRet * 0x100) + BuffRecEF[7].ValRet;
            AffDmap(Tmp16a);
          }
        }
      }
    }
  }
}
```

```

    }
  }
  else { ErreurCmd(); }
}
else {Lcd.print(MessLcd[13]);}
}
break;

```

```

case 7: { // ----- R  cup  ration infos AF B2
  if (InitCmd()==0) {break;}
  RazBuff_EF();
  BuffSendEF[1]=0x0A;
  BuffSendEF[3]=0xB2;
  SendBuff_EF(8);
  if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
    Lcd.print("Oct1a3: ");
    LcdHex8(BuffRecEF[4].ValRet);
    Lcd.setCursor(11,1);
    LcdHex8(BuffRecEF[5].ValRet);
    Lcd.setCursor(14,1);
    LcdHex8(BuffRecEF[6].ValRet);
  }
  else { ErreurCmd(); }
}
break;

```

```

case 8: { // ----- Lens extension factor E4
  if (InitCmd()==0) {break;}
  Cmd_Aff2x(0xE4);
}
break;

```

```

case 17: { // ----- Registre AF 6octets E8
  if (InitCmd()==0) {break;}
  Cmd_Aff6x(0xE8);
}
break;

```

```

case 18: { // ----- Registre AF 4octets EA
  if (InitCmd()==0) {break;}
  Cmd_Aff6x(0xEA);
}
break;

```

```

case 19: { // ----- R  cup  ration valeur codeur AF (C4)
  if (InitCmd()==0) {break;}
  Cmd_Aff2x(0xC4);
}
break;

```

```

case 20: { // ----- Statut stabilisation
  if ((EF_Fct1 & 0x10) == 0x10) {
    if (InitCmd()==0) {break;}
    if (millis() - CptStab > TempoStab) {Tmp8a=0x99;}
    else {Tmp8a=0xB9;}
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0x91;
    BuffSendEF[4]=Tmp8a;
    SendBuff_EF(8);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
      if(BuffRecEF[6].ValRet==0) {
        Lcd.print(MessLcd[21]);
        CptStab=0;}
      else if(BuffRecEF[6].ValRet==0x04) {Lcd.print(MessLcd[22]);}
      else if(BuffRecEF[6].ValRet==0x0C) {Lcd.print(MessLcd[24]);}
      else if(BuffRecEF[6].ValRet==0x0E) {Lcd.print(MessLcd[23]);}
      else if(BuffRecEF[6].ValRet==0x1C) {Lcd.print(MessLcd[25]);}
      else {
        Lcd.print(MessLcd[26]);
        Lcd.setCursor(11,1);
        LcdHex8(BuffRecEF[6].ValRet);
      }
    }
    else { ErreurCmd(); }
  }
  else {Lcd.print(MessLcd[20]);}
}
break;

```

```

} //FinSwitch
} //Fin void

```

```
// =====
// = Tests exécutés sur appui touches flèche droite ou gauche
// =====

void Test_KeyH(uint8_t Sens) {

  Lcd.setCursor(0,1);

  switch (OrdreMenu[IdxMenu]) {

    case 9: { // ----- Iris min/max
      if (InitCmd()==0) {break;}
      Cmd_B0();
      if (EF_AvT!=0) {
        if (Sens==1) {Tmp8a=EF_AvMin-EF_AvMax;}
        else {Tmp8a=0x80;}
        Tmp16a=millis();
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x00;
        BuffSendEF[5]=0x13;
        BuffSendEF[6]=Tmp8a;
        SendBuff_EF(8);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
          if ((Cmd_90() & 0x01) !=Sens) {Lcd.print(MessLcd[19]);}
          else {
            Tmp16a=millis()- Tmp16a;
            Lcd.print(MessLcd[18]);
            Lcd.setCursor(10,1);
            Lcd.print(Tmp16a);
            Lcd.print("ms");
          }
        }
        else { ErreurCmd(); }
      }
    }
    break;

    case 10: { // ----- Iris continu
      if (InitCmd()==0) {break;}
      Cmd_B0();
      if (EF_AvT!=0) {
        if (Sens==0 || EF_CptAv < EF_AvMin) {
          if (Sens==1) {Tmp8a=0x01;}

```

```

        else {Tmp8a=0xFF;}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x00;
        BuffSendEF[5]=0x13;
        BuffSendEF[6]=Tmp8a;
        SendBuff_EF(8);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
          Lcd.print(MessLcd[17]);
          Lcd.setCursor(13,1);
          if ((Cmd_90() & 0x01) ==0) {
            EF_CptAv=EF_AvMax;
            Lcd.print("PO");
          }
          else if (EF_CptAv!=0) {
            if (Sens==1) {EF_CptAv++;}
            else {EF_CptAv--;}
            Lcd.print(EF_CptAv);
          }
        }
        else { ErreurCmd(); }
      }
    }
    break;

    case 11: { // ----- AF mini / infini
      if (InitCmd()==0) {break;}
      if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}
      else {
        Tmp16a=millis();
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x0C;
        BuffSendEF[5]=0x50;
        BuffSendEF[6]=VitesseAF[IdxVitesseAF];
        BuffSendEF[7]=05+Sens;
        SendBuff_EF(10);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
          for (Cpt16=0; (Cpt16 < 0x200) && ((Cmd_90() & 0x24) != 0); Cpt16++) {delay(2);}
          if (Cpt16 == 0x200) { Lcd.print(MessLcd[16]);}
          else {
            Tmp16a=millis()- Tmp16a;
            Lcd.print(MessLcd[15]);
            Lcd.setCursor(9,1);
            Lcd.print(Tmp16a);

```

```

        Lcd.print("ms");
    }
}
else { ErreurCmd(); }
}
}
break;

case 12: { // ----- AF Continu C0
    if (InitCmd()==0) {break;}
    if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}
    else {
        if (Sens==0) {Tmp8a=0x00; Tmp8b=0x10;}
        else {Tmp8a=0xFF; Tmp8b=0xF0;}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x50;
        BuffSendEF[4]=VitesseAF[IdxVitesseAF];
        BuffSendEF[5]=0x44;
        BuffSendEF[6]=Tmp8a;
        BuffSendEF[7]=Tmp8b;
        SendBuff_EF(9);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            for (Cpt16=0; (Cpt16 < 0x200) && ((Cmd_90() & 0x24) != 0); Cpt16++) {delay(2);}
            if (Cpt16 == 0x200) { Lcd.print(MessLcd[16]);}
            else {
                Lcd.print(MessLcd[15]);
                if ((BuffRecEF[4].ValRet & 0x10) == 0x10) {
                    Lcd.setCursor(9,1);
                    Lcd.print("/");}
                RazBuff_EF();
                BuffSendEF[1]=0x0A;
                BuffSendEF[3]=0xC0;
                SendBuff_EF(7);
                Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
                Lcd.setCursor(11,1);
                Lcd.print(Tmp16a,HEX);
            }
        }
        else { ErreurCmd(); }
    }
}
break;

case 13: { // ----- AF Continu C2
    if (InitCmd()==0) {break;}

```

```

    if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}
    else {
        if (Sens==0) {Tmp8a=0x00; Tmp8b=0x10;}
        else {Tmp8a=0xFF; Tmp8b=0xF0;}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x0C;
        BuffSendEF[5]=0x50;
        BuffSendEF[6]=VitesseAF[IdxVitesseAF];
        BuffSendEF[7]=0x44;
        BuffSendEF[8]=Tmp8a;
        BuffSendEF[9]=Tmp8b;
        BuffSendEF[10]=0xC2;
        SendBuff_EF(14);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            Tmp16a=(BuffRecEF[11].ValRet * 0x100) + BuffRecEF[12].ValRet;
            for (Cpt16=0; (Cpt16 < 0x200) && ((Cmd_90() & 0x24) != 0); Cpt16++) {delay(2);}
            if (Cpt16 == 0x200) { Lcd.print(MessLcd[16]);}
            else {
                Lcd.print(MessLcd[15]);
                if ((BuffRecEF[4].ValRet & 0x10) == 0x10) {
                    Lcd.setCursor(9,1);
                    Lcd.print("/");}
                Lcd.setCursor(10,1);
                if (Tmp16a > 9999) {Lcd.print("inf ");}
                else {
                    Lcd.print(Tmp16a);
                    Lcd.print("cm");
                }
            }
        }
        else { ErreurCmd(); }
    }
}
break;

case 14: { // ----- Modification vitesse AF
    if (Sens==1) {
        if (IdxVitesseAF < 5) {IdxVitesseAF++;}
    }
    else {
        if (IdxVitesseAF > 0){IdxVitesseAF--;}
    }
    AffVitAF();
}
break;

```

```

case 15: { // ----- Eclairage objectif
    digitalWrite (LedLens,Sens);
    Lcd.print (MessLcd[8+Sens]);
}
break;

```

```

case 16: { // ----- Debug On/Off
    FlagDebug=Sens;
    Lcd.print (MessLcd[6+Sens]);
}
break;

} //FinSwitch
} //Fin void

```

Test_Select.ino

```

// =====
// =          Tests exécutés sur appui touche Select
// =====

```

```

void Test_Select() {

    Lcd.setCursor(0,1);

    switch (OrdreMenu[IdxMenu]) {

        case 00: { // ----- Insertion
            if (EF_AlimOk==1) {Arret_EF();}
            else {
                Init_EF();
                if (EF_InitOk==1) {Lcd.print(MessLcd[4]);}
            }
        }
        break;

        case 10: { // ----- Iris a PO
            if (InitCmd()==0) {break;}
            Cmd_13PO();
        }
        break;

```

```

case 12: { // ----- Raz Compteur AF
    if (InitCmd()==0) {break;}
    if ((Cmd_90() & 0x80)== 0x80) { Lcd.print(MessLcd[14]);}
    else {
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x0C;
        BuffSendEF[5]=0xC0;
        SendBuff_EF(9);
        Tmp16a=(BuffRecEF[6].ValRet * 0x100) + BuffRecEF[7].ValRet;
        Lcd.setCursor(11,1);
        Lcd.print(Tmp16a,HEX);
        Lcd.print(" ");
    }
}
break;

case 20: { // ----- IS On
    CptStab=millis();
}
break;

} //FinSwitch
} //Fin void

```

```
// =====
// = Tests exécutés une seule fois a chaque changement de menu
// =====

void Test_Solo() {

  Lcd.setCursor(0,1);

  switch (OrdreMenu[IdxMenu]) {

    case 00: { // -----Insertion;
      Lcd.print (MessLcd[2+EF_AlimOk]);
    }
    break;

    case 01: { // -----Récupération paramètres
      if (InitCmd()==0) {break;}
      Cmd_80();
      if (EF_IdLens!=0) {
        for (Cpt=4; Cpt < 12; Cpt++) { LcdHex8(BuffRecEF[Cpt].ValRet); }
        Serial.println(NomMenu[OrdreMenu[IdxMenu]]);
        Serial.print(" Lens Id number = ");
        Serial.println(EF_IdLens);
        Serial.print(" Focale min = ");
        Serial.print(EF_Fmin);
        Serial.println ("mm");
        Serial.print(" Focale max = ");
        Serial.print(EF_Fmax);
        Serial.println ("mm");
        Serial.print(" Fonction1 = ");
        PrintHexBin8(EF_Fct1);
        Serial.print(" Fonction2 = ");
        PrintHexBin8(EF_Fct2);
        if ((EF_Fct2 & 0x80) == 0x80) {
          Serial.print(" Extension ");
          for (Cpt=12; Cpt < 16; Cpt++) { PrintHex8(BuffRecEF[Cpt].ValRet); }
          Serial.println();
        }
        Serial.println();
      }
    }
    break;
  }
}
```

```
case 02: { // -----Récupération nom objectif
  if (InitCmd()==0) {break;}
  if ((EF_Fct2 & 0x80) == 0x80) {
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0x82;
    for (Cpt=4; Cpt < 32; Cpt++) {BuffSendEF[Cpt]=0x83;}
    SendBuff_EF(32);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
      if (BuffRecEF[4].ValRet==0x82) {Lcd.print(MessLcd[12]);}
      else {
        for (Cpt=4; Cpt<20 || BuffRecEF[Cpt].ValRet==0; Cpt++) {Lcd.write(BuffRecEF[Cpt].ValRet);}
        Serial.print(NomMenu[OrdreMenu[IdxMenu]]);
        Serial.print(" : ");
        for (Cpt=4; Cpt<32 || BuffRecEF[Cpt].ValRet==0; Cpt++) {Serial.write(BuffRecEF[Cpt].ValRet);}
        Serial.println();
      }
    }
    else { ErreurCmd(); }
  }
  else {Lcd.print(MessLcd[13]);}
}
break;

case 10: { // -----Iris a "PO
  if (InitCmd()==0) {break;}
  Cmd_13PO();
}
break;

case 12: { // -----Aff Compteur AF
  if (InitCmd()==0) {break;}
  if ((Cmd_90() & 0x80)== 0x80) { Lcd.print(MessLcd[14]);}
  else {
    Lcd.print(MessLcd[15]);
    if ((Cmd_90() & 0x10) == 0x10) {
      Lcd.setCursor(9,1);
      Lcd.print("/");
    }
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0xC0;
    SendBuff_EF(7);
  }
}
```

```

    Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
    Lcd.setCursor(11,1);
    Lcd.print(Tmp16a,HEX);
}
}
break;

```

```

case 13: { // ----- Aff d map
    if (InitCmd()==0) {break;}
    if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}
    else {
        Lcd.print(MessLcd[15]);
        if ((Cmd_90() & 0x10) == 0x10) {
            Lcd.setCursor(9,1);
            Lcd.print("/");}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0xC2;
        SendBuff_EF(7);
        Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
        Lcd.setCursor(10,1);
        if (Tmp16a > 9999) {Lcd.print(" inf ");}
        else {
            Lcd.print(Tmp16a);
            Lcd.print("cm"); }
    }
}
break;

```

```

case 14: { // -----Vitesse AF
    AffVitAF();
}
break;

```

```

case 15: { // ----- Eclairage objectif
    Tmp8a=digitalRead(LedLens);
    Lcd.print (MessLcd[8+Tmp8a]);
}
break;

```

```

case 16: { // ----- Debug On/Off
    Lcd.print (MessLcd[6+FlagDebug]);
}
break;

```

```

}
}

```

SPIxEos.h

```

struct SPIEOS_Ret
{
    uint8_t ValRet;
    uint16_t tRep;
};

```

```

/==== Envoi et réception 8bits en mode SPI Motorola_3 + contrôle temps réponse objectif Canon EOS ====
//=
//= Version minimale sans fonction retard, Temps Horloge0 = 170ns, H1= 500ns, Horloge 1.4Mhz env. =
//=
//= Octet envoyé = Valeur passé a la fonction au format int8_t =
//= Retour fonction dans variable au format Structure SPIEOS_Ret : Octet reçu dans .ValRet, durée signal =
//= de validation objectif dans .tRep ou 0xFFFF si pas de signal, ou 0xFFFF si dépassement de durée =
//=
//=====

```

```

//Version transmission basse vitesse protocole EOS de base -----
// t Horloge0=5.9µs , tH1=6.5µs, Horloge= 80.3kHz

```

```

SPIEOS_Ret SPILSend (uint8_t DataSend)
{
    uint8_t CptBit;
    uint8_t ValPort;
    uint8_t Retmp=0;
    SPIEOS_Ret ValRet;

    for (CptBit=0; CptBit < 8; CptBit++)
    {
        ValPort=(DataSend & 0x80) | 0x3F; // Horloge a 0, DCL positionné
        PORTA=ValPort;
        DataSend = DataSend <<1;
        Retmp=Retmp <<1;
        delayMicroseconds(7);

        PORTA=ValPort | 0x7F; // Horloge a 1, DCL positionné
        Retmp=Retmp | (PINA & 0x01);
        delayMicroseconds(7);
    }

    delayMicroseconds(2); //Remise a 1 Data et Horloge
    PORTA=0xFF;
    ValRet.ValRet=Retmp;
    ValRet.tRep=SPIEOS_tValid();
    return(ValRet);
}

```

```

//Version transmission basse vitesse protocole EOS de base -----
// t Horloge0=0.916µs , tH1=1.2µs, Horloge= 470kHz

```

```

SPIEOS_Ret SPIHSend (uint8_t DataSend)
{
    uint8_t CptBit;
    uint8_t ValPort;
    uint8_t Retmp=0;
    SPIEOS_Ret ValRet;
    volatile uint8_t DummyH=0; //Pour instructions bidon d'ajustement temps

    noInterrupts();
    for (CptBit=0; CptBit < 8; CptBit++)
    {
        ValPort=(DataSend & 0x80) | 0x3F; // Horloge a 0, DCL positionné
        PORTA=ValPort;
        DataSend = DataSend <<1;
        Retmp=Retmp <<1;
        DummyH=PINA;
        DummyH=DummyH >> 2;

        PORTA=ValPort | 0x7F; // Horloge a 1, DCL positionné
        Retmp=Retmp | (PINA & 0x01);
        DummyH=DummyH >> 1;
    }
    interrupts();

    delayMicroseconds(2); //Remise a 1 Data et Horloge
    PORTA=0xFF;
    ValRet.ValRet=Retmp;
    ValRet.tRep=SPIEOS_tValid();
    return(ValRet);
}

```

```

//Calcul durée signal validation-occupation niveau bas horloge objectif -----
//Retour = Durée signal, 0xFFFF si pas de validation, 0xFFFF si dépassement durée.

```

```

uint16_t SPIEOS_tValid()
{
    uint16_t Cpt;
    uint8_t Test;

    Test=(PINA & 0x02);
    Cpt=0x0000;

```

```

while (Test==2) { //Attente passage horloge au niveau bas
Test=(PINA & 0x02);
Cpt++;
if (Cpt > 0x027) { //Si pas de reponse avant 30ms => valeur de retour = 0xFFFFE
Cpt=0xFFFFE; //Valeur limite : 0x16=16µs, 0x20=28µs, 0x27=33µs,
Test=00;
}
}

if (Cpt < 0xFF00) {
Cpt=0x0000; //Attente passage horloge au niveau haut
Test=(PINA & 0x02);
while (Test==0){
Test=(PINA & 0x02);
Cpt++;
if (Cpt > 0xFFFF0) { //Si tps low supérieur a limite => valeur de retour = 0xFFFF
Cpt=0xFFFF;
Test=02;
}
delayMicroseconds(3); //Pour compteur =FFF0 t(1))62ms, t(2)=107ms, t(3)=173.3ms (1 unité=2.8µs
env.), t(4)=239ms
}
}
return (Cpt);
}

```

Version II



Interface Physique de test

La conception de l'interface entre la carte CPU Mega et l'objectif est entièrement repensée pour obtenir une solution plus pratique et plus sûre d'emploi. L'électronique de l'interface est contenue sur un petit circuit imprimé s'enfichant directement sur le connecteur 36points de la carte Mega à l'instar d'un shield standard. L'adaptateur objectif est relié à cette interface par un câble en nappe d'une cinquantaine de centimètres de long permettant ainsi de manipuler cet objectif facilement en cours de test.

D'un point de vue électronique le principal point de différence avec le modèle précédent réside dans l'utilisation de buffers à collecteur ouvert pour commander les entrées data et horloge de l'objectif. Des résistances de tirage au +5v supplémentaire d'une valeur de 1.5k ohms sont utilisées pour limiter le temps de de-saturation tPLH des transistors.

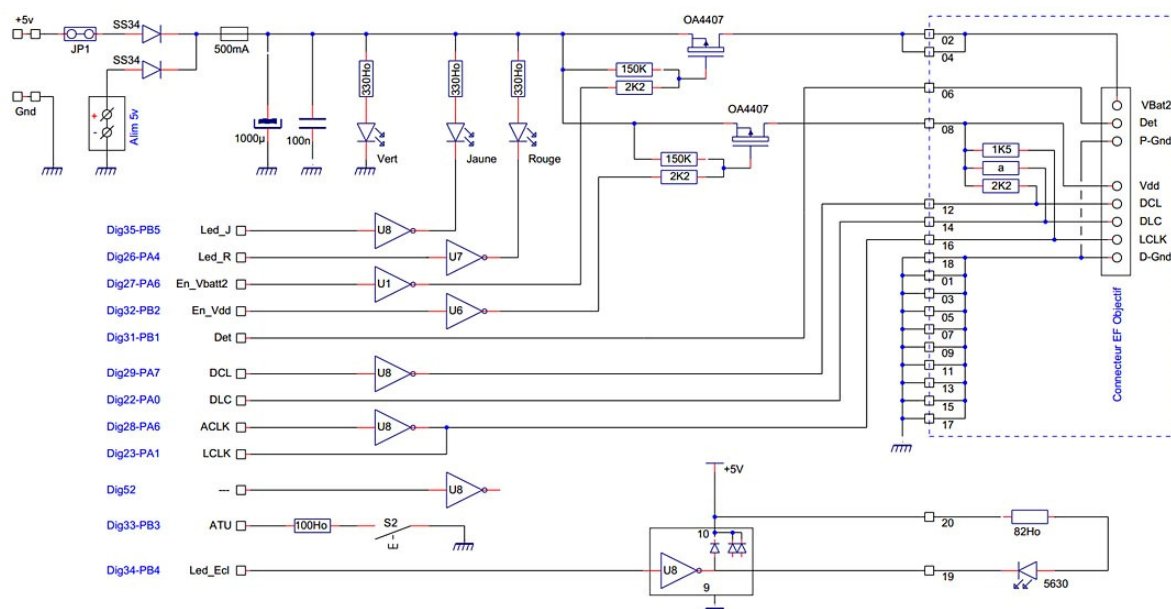
Les deux alimentations 5v de l'objectif (CPU et Moteur) sont pilotées par le système et la broche de détection objectif du connecteur EF utilisée à cet effet. Ces tensions seront donc désactivées soit automatiquement par le logiciel à la fin de la période de veille, soit lors de l'appui sur le poussoir d'arrêt de la carte interface, soit lors du retrait de l'objectif sur son connecteur

Si des objectifs légers avec un moteur AF récent peuvent être alimentés directement par la carte CPU il sera préférable d'utiliser une alimentation externe dans les autres cas (objectifs lourds ou dotés d'IS). Pour cela une prise USB femelle de type B est implantée sur la carte interface, si deux diodes anti-retour sont présentes pour éviter les échanges de courant entre les sources externe et interne - CPU Arduino un jumper permet de désactiver et d'isoler cette dernière complètement.

Trois Leds de signalisation sont présentes sur la carte interface avec les fonctions suivantes :

- Verte : Présence tension 5v sur la carte interface.
- Rouge : Alimentation Vbatt2 de l'objectif validée.
- Jaune : Dialogue entre la carte CPU et l'objectif en cours.

Schéma interface



Connecteurs interface

Connecteur Arduino MEGA

		+5v	+5v	
Data in - DLC	PA0	22	23	PA1
	PA2	24	25	PA3
Led En_Vbatt2	PA4	26	27	PA5
Clock out - SKC	PA6	28	29	PA7
	Dig 30	30	31	Dig 31
(5v CPU) En_Vdd	Dig 32	32	33	Dig 33
En_Led éclairage	Dig 34	34	35	Dig 35
(Non utilisé) U8		52	53	
(Vert) Gnd		0v	0v	

LCLK - Clock in
En_Vbatt2 (5v mot)
DCL - Data out
Détection objectif
Poussoir arrêt
Led jaune

Led Diag +
Led Diag -

Connecteur adaptateur EF

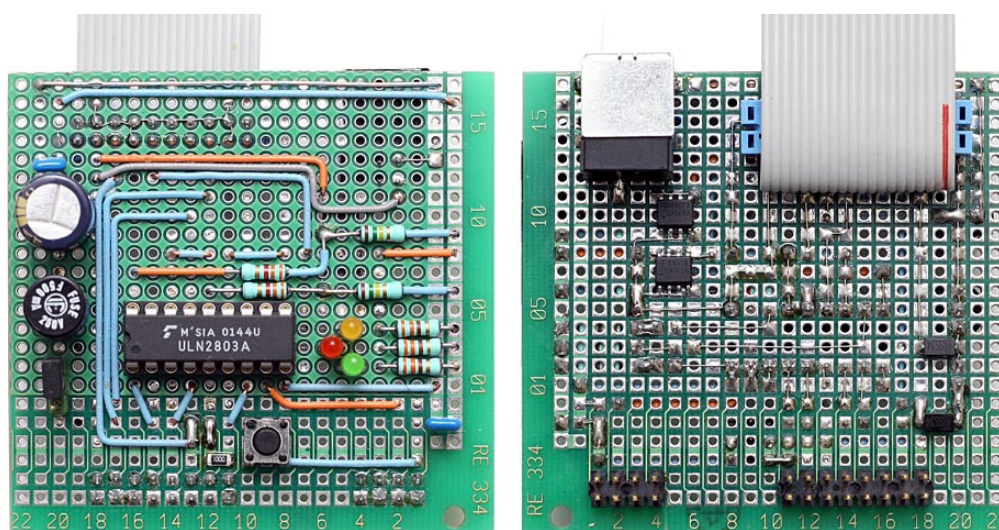
(HE10 male vu de dessus)

Led éclairage +	20	19	Led éclairage -
Gnd	18	17	Gnd
Clock	16	15	Gnd
(Data out) DLC	14	13	Gnd
(Data in) DCL	12	11	Gnd
(5v cpu) Vdd	10	9	Gnd
Gnd	8	7	Gnd
Det objectif	6	5	Gnd
Vbatt2	4	3	Gnd
(5v Mot.) Vbatt2	2	1	Gnd

Interface Arduino

Si la conception d'un circuit imprimé gravé ne pose pas de grandes difficultés en DIY le pas de montage des composants étant de 2.54mm une plaque d'expérimentation standard est encore utilisée pour réaliser ce petit prototype.

Deux connecteur 20 points auto-sertissants sont utilisés pour relier la nappe de fil, un de type HE10 femelle, l'autre à souder. Ce dernier et la prise USB sont montés sur le coté inverse du circuit imprimé pour servir de support et éviter les efforts de flexion au niveau du connecteur Arduino. Pour faciliter le câblage de la plaque d'expérimentation la nappe est montée à l'envers et retournée pour une sortie du bon coté.



Adaptateur EF

Une ancienne bague allonge Kenko hors service est utilisée, un morceau de plaque plastique de 2mm d'épaisseur vient boucher la monture coté boîtier. Le connecteur HE10 male a verrouillage est fixé dessus deux vis venant solidariser le tout avec la partie métallique de la monture. Le câblage entre le connecteur et les broches de la bague allonge est réalisé directement avec du fil rigide 35swg un adhésif isolant assurant leur tenue. Les 3 résistances de tirage et la Led blanche d'éclairage sont aussi câblées en direct sur le substrat plastique.



Structure programme

Le logiciel précédent est repris presque intégralement, seuls les points et modules suivants subissent quelques modification ou ajouts :

Fichier principal :

- Définition des ports E/S, ajout entrée EF_Det et sortie commande Led jaune Led SPI.
- Modification quelques noms VccEF => EF_EnVdd, LedLens => Led_Lens
- Ajout flag de gestion de la vitesse de transfert a 80 ou 500kHz : EF_VitesseSPI.
- Modification des valeurs du port A au repos.
- Ajout gestion de l'entrée EF_Det et d'extinction de l'objectif en cas de retrait.

Fonctions.Ino :

- Modification des valeurs de commande du port A les signaux DCL et Clock_Arduino étant inversés par l'utilisation d'un transistor en CO.
- Déplacement du contrôle de la présence de tension d'alimentation de la fonction InitCmd() a la fonction SendBuff_EF().
- Modification fonction SendBuf_EF() pour le choix de la vitesse d'émission.

SPIxEOS.ino :

- Modification des valeurs de commande du port A les signaux DCL et Clock_Arduino étant inversés par l'utilisation d'un transistor en CO.

TestKeyH.ino et Solo.ino

- Ajout gestion du choix de la vitesse de dialogue du port EF à 80 ou 500kHz.

TestAuto.ino

- Ajout affichage registres BestFocus Adjustment F8 a FE.

Listing complet

Le listing qui suit n'est donné qu'à titre indicatif dans le cas d'un Pdf orphelin, seule la dernière version disponible ici sera à prendre en compte.

Version en cours : 2.01 => http://jp79dsfr.free.fr/_Docs%20et%20infos/Sources/Protocole%20EF/Sketchs/Canon_Test_objectif_v2.zip

Boucle principale

```
// =====
//=
//=      Logiciel de diagnostic et test du fonctionnement des objectifs EOS Canon
//=
//=  Chargement sur platine Mega uniquement avec shield LCD-Boutons standard
//=                                     © JP 2016 =
// =====

//===== Déclarations
#include <LiquidCrystal.h>
#include "SPIxEOS.h"

const char* Version="v2.00";

LiquidCrystal Lcd(8,9,4,5,6,7); //----- Définition paramètres shield clavier - LCD
const char LcdNull="";
const uint8_t PortAnKey = 0;
uint16_t AnalogKey = 0;
uint8_t ValKey = 0;
uint8_t MemoKey = 0x20;
const uint8_t KeyHaut =0x5;
const uint8_t KeyBas =0xA;
const uint8_t KeyDroit =0x1;
const uint8_t KeyGauche =0xF;
const uint8_t KeySel =0x17;
const uint8_t KeyNone =0x20;

const uint8_t Led_CDG = 53; //----- Définitions ports E/S
const uint8_t Led_Test= 52;
const uint8_t Led_SPI= 35;
const uint8_t Led_Lens= 34;
const uint8_t EF_Det=31;
const uint8_t EF_EnVdd= 32;

const uint8_t Bp_ATU=33;
const uint8_t Bp_Debug=21;
//Interface EF sur PortA, modification logiciel si changement

uint8_t CliLedCDG=0; //----- Variables temporisations cycle et chien de garde
uint32_t CptLedCDG;
uint8_t CliLedTest=0;

const uint16_t RepeatAuto=200; //Intervalle entre test Auto en ms
uint32_t CptTempoAuto;

const uint8_t RepeatKeyL=25; //Intervalle avant début répétition touche x 25ms
const uint8_t RepeatKeyH=2; //Intervalle entre répétitions touche x 25ms
uint32_t CptTempoKey;
uint8_t CptRepeatKey=1;
uint8_t RepeatKey=RepeatKeyL;

const uint32_t TempoAlimEF= 180000; //Durée maintient alim objectif apres dernier envoi
uint32_t CptAlimEF;
const uint16_t TempoAlimMot= 500; //Durée maintient Vbatt2 apres dernier envoi
uint32_t CptAlimMot;
const uint16_t TempoStab= 1000; //Durée minimale action Is
uint32_t CptStab;
uint32_t ValTemps;

//----- Gestion et définitions des menus
const uint8_t MaxMenu=21; //Nb de menu max -1
uint8_t IdxMenu=0; //Index Menu en cours (de 0 a MaxMenu)
//Ordre d'affichage menus

const uint8_t OrdreMenu[22]={0,1,2,3,4,6,7,5,8,19,17,18,9,10,11,12,13,20,14,21,15,16}; const char*
NomMenu[23]={
  " Insertion",
  "Caracteristiques",
  " Nom objectif",
  " Av Min/Max",
```

```

"Focale actuelle",
"Codeur Zoom  E0",
"Distance map  C2",
"Best AF adj.  B2",
"Lens ExtFact. E4",
" Cmd AV MAX/min",
" Cmd AV continu",
"Cmd AF mini/inf.",
"Cmd. AF Lin - C0",
"Cmd. AF Lin - C2",
"Adj. Vitesse AF",
" Eclairage EF",
" Trace SPI-EF",
"Registre AF  E8",
"Registre AF  EA",
"Codeur AF   C4",
" Commande IS",
"Vitesse dialogue",
"";

```

```

const char* MessLcd[32] ={
"Banc test EOS-EF",
"Retrait objectif",
" Tension Off  ",
" Tension ON  ",
"Init objectif Ok",
"Pas de reponse ",
"Serial debug Off",
"Serial debug On",
" Led EF : Off  ",
" Led EF : On  ",
" Vitesse AF=$  ",
"Erreur reception",
" Cmd inconnue  ",
" Non disponible ",
" AF en manuel  ",
" AF Ok :      ",
" Erreur AF    ",
" Valeur iris:  ",
" Tps fct:     ",
"Erreur switch PO",
"Objectif sans IS",
" Inter IS Off  ",
" IS a l'arret  ",
" IS en fonction ",
" Initialisation ",
"(de)Verrouillage",
" Statut=      ",

```

```

" Init  ",
" Reset Obj.  ",
" Bus SPI: 80kHz",
" Bus SPI:500kHz ",
"";

```

```
// ----- Valeurs Dialogue EF
```

```

uint8_t EF_InitOk=0;           //Flag optique présente
uint8_t EF_AlimOk=0;           //Flag présence alim Vcc 5v objectif
uint8_t EF_VitesseSPI=0;       //Flag vitesse bus SPI 0=80kHz _ 1=500kHz

```

```
//Memo caractéristiques Lens
```

```

uint8_t EF_TypeLens;
uint8_t EF_IdLens;
uint16_t EF_Fmin;
uint16_t EF_Fmax;
uint8_t EF_Fct1;
uint8_t EF_Fct2;
uint8_t EF_AvT;
uint8_t EF_AvMin;
uint8_t EF_AvMax;
uint8_t EF_CptAv;

```

```

uint8_t IdxVitesseAF=3;        //Tables valeurs vitesse moteur AF
uint8_t VitesseAF[6]={0,0x2C, 0x2D, 0x2E, 0x2F, 0xFF};

```

```

uint8_t BuffSendEF[32];        //Buffer Emission réception
SPIEOS_Ret BuffRecEF[32];

```

```

uint8_t FlagDebug=0;           // Pour test et debug comm EF
uint8_t ValChange8;            // Pour memo variation de valeurs

```

```

uint16_t ValChange16;
uint8_t Tmp8a;                 // Variables temporaires et de travail
uint8_t Tmp8b;
uint16_t Tmp16a;
uint8_t Cpt;
uint16_t Cpt16;

```

```
// ===== Initialisation
```

```

void setup() {
  Lcd.begin (16,2);
  Lcd.clear();
  Lcd.setCursor(0,0);
  Lcd.print(MessLcd[0]);
  Lcd.setCursor(5,1);
  Lcd.print(Version);

```

```

DDRA=0x0F;                     //Init entrées sorties
PORTA=0x0F;

```

```

pinMode(Led_CDG, OUTPUT);
digitalWrite(Led_CDG, LOW);
pinMode(Led_Test, OUTPUT);
digitalWrite(Led_Test, LOW);
pinMode(Led_SPI, OUTPUT);
digitalWrite(Led_SPI, LOW);
pinMode(Led_Lens, OUTPUT);
digitalWrite(Led_Lens, LOW);
pinMode(EF_EnVdd, OUTPUT);
digitalWrite(EF_EnVdd, LOW);
pinMode (EF_Det, INPUT);
digitalWrite(EF_Det, HIGH);
pinMode (Bp_ATU, INPUT);
digitalWrite(Bp_ATU, HIGH);
pinMode (Bp_Debug, INPUT);
digitalWrite(Bp_Debug, HIGH);

Serial.begin(115200);
Serial.print("\r\nLogiciel de Test du fonctionnement des objectifs EOS Canon _ ");
Serial.print(Versions);
Serial.print("\r\n\r\n");

delay (2500);
CptAlimEF=0xFFFFF;
CptAlimMot=0xFFFFF;
CptStab=0xFFFFF;
RefreshMenu();
RazData_EF();
}

// ===== Boucle principale

void loop() {

ValTemps=millis();

if (ValTemps-CptLcdCDG > 750){ // ----- Chien de garde
  CliLcdCDG = !CliLcdCDG;
  digitalWrite(Led_CDG, CliLcdCDG);
  CptLcdCDG=ValTemps;
}

if (ValTemps - CptAlimMot > TempoAlimMot) { // ----- Coupure auto Vbatt2 après envoi
  PORTA=0x0F;
}

if (EF_AlumOk==1) {
if (ValTemps - CptAlimEF > TempoAlimEF) { // ----- Extinction auto interface EF après touche
  Arret_EF();

```

```

}}

if (ValTemps-CptTempoKey > 25){ // ----- Gestion des Touches
  CptTempoKey=ValTemps;
  AnalogKey = analogRead(PortAnKey);
  ValKey = (AnalogKey / 32) + 1;

  if (MemoKey != ValKey){ //Changement de touche ou repetition
    MemoKey = ValKey;
    CptAlimEF=ValTemps;

    if (ValKey == KeyHaut) {
      if (IdxMenu==0) {IdxMenu=MaxMenu;}
      else {IdxMenu--;}
      RefreshMenu();
    }

    if (ValKey == KeyBas) {
      if (IdxMenu==MaxMenu) {IdxMenu=0;}
      else {IdxMenu++;}
      RefreshMenu();
    }

    if (ValKey == KeySel) {Test_Select();}

    if (ValKey == KeyGauche) {Test_KeyH(0);}
    if (ValKey == KeyDroit) {Test_KeyH(1);}
  }

  else { // ----- Répétition touches
    if (ValKey == KeyNone) {
      RepeatKey = RepeatKeyL;
      CptRepeatKey=1; }
    else {
      CptRepeatKey ++;
      if (CptRepeatKey == RepeatKey) {
        MemoKey = KeyNone;
        CptRepeatKey=1;
        RepeatKey = RepeatKeyH; }
    }

    if (EF_AlumOk ==1) { // ----- Bouton arrêt d'urgence
      Tmp8b=digitalRead(Bp_ATU);
      if ( Tmp8b==LOW ) {Arret_EF();}
    }
    Tmp8a=digitalRead(Bp_Debug); // ----- Bouton debug
    if (Tmp8a==LOW) {Debug_A();}
  }
}

```

```

if (EF_InitOk ==1){ // ----- Tests automatiques
if (ValTemps-CptTempoAuto > RepeatAuto){
    CptTempoAuto=ValTemps;
    Test_Auto();
}
}

```

```

if (EF_AlimOk ==1) { // ----- Entrée détection objectif
    Tmp8b=digitalRead(EF_Det);
    if ( Tmp8b==HIGH ) {
        PORTA=0x0F;
        digitalWrite(EF_EnVdd,LOW);
        Lcd.setCursor(0,1);
        Lcd.print (MessLcd[01]);
        Serial.println ("Retrait objectif");
        delay (2500);
        Arret_EF(); }
    }
}

```

Cmd_EF

```

// =====
// = Commandes envoyées a l'objectif
// =====

```

```

void Cmd_13PO() { // ----- Ouverture Iris a PO
    Cmd_B0();
    if (EF_AvT!=0) {
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x13;
        BuffSendEF[4]=0x80;
        SendBuff_EF(8);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            Lcd.print(MessLcd[17]);
            Lcd.setCursor(13,1);
            if ((Cmd_90()& 0x01) ==0) {
                EF_CptAv=EF_AvMax;
                Lcd.print("PO");
            }
        }
        else { ErreurCmd(); }
    }
}

```

```

void Cmd_80(){ // ----- Récupération paramètres objectif
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0x80;
    BuffSendEF[4]=0x0A;

```

```

    BuffSendEF[5]=0x99; //Emulation 40D
    BuffSendEF[6]=0x01;
    SendBuff_EF(17);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        EF_TypeLens=BuffRecEF[4].ValRet;
        EF_IdLens=BuffRecEF[5].ValRet;
        EF_Fmin=(BuffRecEF[6].ValRet * 0x100) + BuffRecEF[7].ValRet;
        EF_Fmax=(BuffRecEF[8].ValRet * 0x100) + BuffRecEF[9].ValRet;
        EF_Fct1=BuffRecEF[10].ValRet;
        EF_Fct2=BuffRecEF[11].ValRet;
    }
    else {
        RazData_EF();
        ErreurCmd();
    }
}

```

```

uint8_t Cmd_90() { // ----- Demande statut AF/AV sans contrôle
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[2]=0x90;
    SendBuff_EF(6);
    return BuffRecEF[4].ValRet;
}

```

```

void Cmd_B0() { // ----- Récupération caractéristiques Av objectif

    RazBuff_EF();
    BuffSendEF[1]=0x0A;

```

```

BuffSendEF[3]=0xB0;
SendBuff_EF(8);
if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
    EF_AvT=BuffRecEF[4].ValRet;
    EF_AvMax=BuffRecEF[5].ValRet;
    EF_AvMin=BuffRecEF[6].ValRet;
}
else {
    EF_AvT=0;
    EF_CptAv=0;
    ErreurCmd(); }
}

void Cmd_Aff2x(uint8_t Val) { // ----- Affiche les 2 octets retournés par Val
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=Val;
    SendBuff_EF(7);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        Lcd.print("Oct1:");
        LcdHex8(BuffRecEF[4].ValRet);
        Lcd.setCursor(9,1);
        Lcd.print("Oct2:");
        LcdHex8(BuffRecEF[5].ValRet);
    }
    else { ErreurCmd(); }
}

```

```

void Cmd_Aff6x(uint8_t Val) { // ----- Affiche les octets retournés par Val
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=Val;
    SendBuff_EF(11);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        Lcd.print(" ");
        LcdHex8(BuffRecEF[4].ValRet);
        LcdHex8(BuffRecEF[5].ValRet);
        Lcd.print("-");
        LcdHex8(BuffRecEF[6].ValRet);
        LcdHex8(BuffRecEF[7].ValRet);
        Lcd.print("-");
        LcdHex8(BuffRecEF[8].ValRet);
        LcdHex8(BuffRecEF[9].ValRet);
    }
    else { ErreurCmd(); }
}

```

Test_Auto.ino

```

// =====
// = Tests exécutés a intervalles réguliers automatiquement
// =====

void Test_Auto() {

    Lcd.setCursor(0,1);

    switch (OrdreMenu[IdxMenu]) {

```

```

case 3: { // ----- Récupération AV min/max (B0)
    if (InitCmd()==0) {break;}
    Cmd_B0();
    if (EF_AvT!=0) {
        Lcd.print(" Max:");
        Lcd.print(EF_AvMax);
        Lcd.print(" Min:");
        Lcd.print(EF_AvMin);
    }
}
break;

```

```

case 4: { // ----- R  cup  ration focale actuelle (A0)
    if (InitCmd()==0) {break;}
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0xA0;
    SendBuff_EF(7);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
        Lcd.setCursor(5,1);
        Lcd.print(Tmp16a);
        Lcd.print("mm ");
    }
    else { ErreurCmd(); }
}
break;

case 5: { // ----- R  cup  ration valeur codeur zoom (E0)
    if (InitCmd()==0) {break;}
    Cmd_Aff2x(0xE0);
}
break;

case 6: { // ----- R  cup  ration distance map r  elle (C2)

    if ((EF_Fct1 & 0x01) == 0x01) {
        if (InitCmd()==0) {break;}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0xC2;
        SendBuff_EF(9);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            if (BuffRecEF[4].ValRet==0xC2) {Lcd.print(MessLcd[12]);}
            else {
                Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
                AffDmap(Tmp16a);
                Lcd.setCursor(8,1);
                Tmp16a=(BuffRecEF[6].ValRet * 0x100) + BuffRecEF[7].ValRet;
                AffDmap(Tmp16a);
            }
        }
        else { ErreurCmd(); }
    }
    else {Lcd.print(MessLcd[13]);}
}
break;

case 7: { // ----- R  cup  ration infos AF (B2)
    if (InitCmd()==0) {break;}
    RazBuff_EF();
    BuffSendEF[1]=0x0A;

```

```

    BuffSendEF[3]=0xB2;
    SendBuff_EF(8);
    if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
        Lcd.print("Oct1a3: ");
        LcdHex8(BuffRecEF[4].ValRet);
        Lcd.setCursor(11,1);
        LcdHex8(BuffRecEF[5].ValRet);
        Lcd.setCursor(14,1);
        LcdHex8(BuffRecEF[6].ValRet);
    }
    else { ErreurCmd(); }
}
break;

```

```

case 8: { // ----- Lens extension factor E4
    if (InitCmd()==0) {break;}
    Cmd_Aff2x(0xE4);
}
break;

```

```

case 17: { // ----- Registre AF 6octets E8
    if (InitCmd()==0) {break;}
    Cmd_Aff6x(0xE8);
}
break;

```

```

case 18: { // ----- Registre AF 4octets EA
    if (InitCmd()==0) {break;}
    Cmd_Aff6x(0xEA);
}
break;

```

```

case 19: { // ----- Recup  ration valeur codeur AF (C4)
    if (InitCmd()==0) {break;}
    Cmd_Aff2x(0xC4);
}
break;

```

```

case 20: { // ----- Statut stabilisation
    if ((EF_Fct1 & 0x10) == 0x10) {
        if (InitCmd()==0) {break;}
        if (millis() - CptStab > TempoStab) {Tmp8a=0x99;}
        else {Tmp8a=0xB9;}
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x91;
        BuffSendEF[4]=Tmp8a;
        SendBuff_EF(8);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            if(BuffRecEF[6].ValRet==0) {

```

```

    Lcd.print(MessLcd[21]);
    CptStab=0;}
else if(BuffRecEF[6].ValRet==0x04) {Lcd.print(MessLcd[22]);}
else if(BuffRecEF[6].ValRet==0x0C) {Lcd.print(MessLcd[24]);}
else if(BuffRecEF[6].ValRet==0x0E) {Lcd.print(MessLcd[23]);}
else if(BuffRecEF[6].ValRet==0x1C) {Lcd.print(MessLcd[25]);}
else {
    Lcd.print(MessLcd[26]);
    Lcd.setCursor(11,1);
    LcdHex8(BuffRecEF[6].ValRet);
}
}

```

```

    else { ErreurCmd(); }
}
else {Lcd.print(MessLcd[20]);}
}
break;

}

}

```

Test_KeyH.ino

```

// =====
// =           Tests exécutés sur appui touches flèche droite ou gauche
// =====

```

```
void Test_KeyH(uint8_t Sens) {
```

```
    Lcd.setCursor(0,1);
```

```
    switch (OrdreMenu[IdxMenu]) {
```

```

    case 9: { // ----- Iris min/max
        if (InitCmd()==0) {break;}
        Cmd_B0();
        if (EF_AvT!=0) {
            if (Sens==1) {Tmp8a=EF_AvMin-EF_AvMax;}
            else {Tmp8a=0x80;}
            Tmp16a=millis();
            RazBuff_EF();
            BuffSendEF[1]=0x0A;
            BuffSendEF[3]=0x00;
            BuffSendEF[5]=0x13;
            BuffSendEF[6]=Tmp8a;
            SendBuff_EF(8);
            if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
                if ((Cmd_90() & 0x01) !=Sens) {Lcd.print(MessLcd[19]);}
            }
            else {

```

```

            Tmp16a=millis()- Tmp16a;
            Lcd.print(MessLcd[18]);
            Lcd.setCursor(10,1);
            Lcd.print(Tmp16a);
            Lcd.print("ms");
        }
    }
    else { ErreurCmd(); }
}
break;

```

```

    case 10: { // ----- Iris continu
        if (InitCmd()==0) {break;}
        Cmd_B0();
        if (EF_AvT!=0) {
            if (Sens==0 || EF_CptAv < EF_AvMin) {
                if (Sens==1) {Tmp8a=0x01;}
                else {Tmp8a=0xFF;}
                RazBuff_EF();
                BuffSendEF[1]=0x0A;
                BuffSendEF[3]=0x00;
                BuffSendEF[5]=0x13;
                BuffSendEF[6]=Tmp8a;
                SendBuff_EF(8);
                if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
                    Lcd.print(MessLcd[17]);
                    Lcd.setCursor(13,1);

```

```

    if ((Cmd_90() & 0x01) == 0) {
        EF_CptAv = EF_AvMax;
        Lcd.print("PO");
    }
    else if (EF_CptAv != 0) {
        if (Sens == 1) { EF_CptAv++; }
        else { EF_CptAv--; }
        Lcd.print(EF_CptAv);
    }
}
else { ErreurCmd(); }
}
}
break;

case 11: { // ----- AF mini / infini
    if (InitCmd() == 0) { break; }
    if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]); }
    else {
        Tmp16a = millis();
        RazBuff_EF();
        BuffSendEF[1] = 0x0A;
        BuffSendEF[3] = 0x0C;
        BuffSendEF[5] = 0x50;
        BuffSendEF[6] = VitesseAF[IdxVitesseAF];
        BuffSendEF[7] = 05 + Sens;
        SendBuff_EF(10);
        if (BuffRecEF[2].ValRet == 0xAA && BuffRecEF[3].ValRet == 0x00) {
            for (Cpt16 = 0; (Cpt16 < 0x200) && ((Cmd_90() & 0x24) != 0); Cpt16++) { delay(2); }
            if (Cpt16 == 0x200) { Lcd.print(MessLcd[16]); }
            else {
                Tmp16a = millis() - Tmp16a;
                Lcd.print(MessLcd[15]);
                Lcd.setCursor(9, 1);
                Lcd.print(Tmp16a);
                Lcd.print("ms");
            }
        }
        else { ErreurCmd(); }
    }
}
break;

case 12: { // ----- AF Continu C0
    if (InitCmd() == 0) { break; }
    if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]); }
    else {
        if (Sens == 0) { Tmp8a = 0x00; Tmp8b = 0x10; }

```

```

    else { Tmp8a = 0xFF; Tmp8b = 0xF0; }
    RazBuff_EF();
    BuffSendEF[1] = 0x0A;
    BuffSendEF[3] = 0x50;
    BuffSendEF[4] = VitesseAF[IdxVitesseAF];
    BuffSendEF[5] = 0x44;
    BuffSendEF[6] = Tmp8a;
    BuffSendEF[7] = Tmp8b;
    SendBuff_EF(9);
    if (BuffRecEF[2].ValRet == 0xAA && BuffRecEF[3].ValRet == 0x00) {
        for (Cpt16 = 0; (Cpt16 < 0x200) && ((Cmd_90() & 0x24) != 0); Cpt16++) { delay(2); }
        if (Cpt16 == 0x200) { Lcd.print(MessLcd[16]); }
        else {
            Lcd.print(MessLcd[15]);
            if ((BuffRecEF[4].ValRet & 0x10) == 0x10) {
                Lcd.setCursor(9, 1);
                Lcd.print("/");
            }
            RazBuff_EF();
            BuffSendEF[1] = 0x0A;
            BuffSendEF[3] = 0x0C;
            SendBuff_EF(7);
            Tmp16a = (BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
            Lcd.setCursor(11, 1);
            Lcd.print(Tmp16a, HEX);
        }
    }
    else { ErreurCmd(); }
}
}
break;

case 13: { // ----- AF Continu C2
    if (InitCmd() == 0) { break; }
    if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]); }
    else {
        if (Sens == 0) { Tmp8a = 0x00; Tmp8b = 0x10; }
        else { Tmp8a = 0xFF; Tmp8b = 0xF0; }
        RazBuff_EF();
        BuffSendEF[1] = 0x0A;
        BuffSendEF[3] = 0x0C;
        BuffSendEF[5] = 0x50;
        BuffSendEF[6] = VitesseAF[IdxVitesseAF];
        BuffSendEF[7] = 0x44;
        BuffSendEF[8] = Tmp8a;
        BuffSendEF[9] = Tmp8b;
        BuffSendEF[10] = 0xC2;
        SendBuff_EF(14);
        if (BuffRecEF[2].ValRet == 0xAA && BuffRecEF[3].ValRet == 0x00) {
            Tmp16a = (BuffRecEF[11].ValRet * 0x100) + BuffRecEF[12].ValRet;

```

```

for (Cpt16=0; (Cpt16 < 0x200) && ((Cmd_90() & 0x24) != 0); Cpt16++) {delay(2);}
if (Cpt16 == 0x200) { Lcd.print(MessLcd[16]);}
else {
  Lcd.print(MessLcd[15]);
  if ((BuffRecEF[4].ValRet & 0x10) == 0x10) {
    Lcd.setCursor(9,1);
    Lcd.print("/");
  }
  Lcd.setCursor(10,1);
  if (Tmp16a > 9999) {Lcd.print("inf ");}
  else {
    Lcd.print(Tmp16a);
    Lcd.print("cm");
  }
}
}
}
else { ErreurCmd(); }
}
}
break;

case 14: { // ----- Modification vitesse AF
  if (Sens==1) {
    if (IdxVitesseAF <5) {IdxVitesseAF++;}
  }
  else {
    if (IdxVitesseAF >0){IdxVitesseAF--;}
  }
}

```

```

  AffVitAF();
}
break;

case 15: { // ----- Eclairage objectif
  digitalWrite (Led_Lens,Sens);
  Lcd.print (MessLcd[8+Sens]);
}
break;

case 16: { // ----- Debug On/Off
  FlagDebug=Sens;
  Lcd.print (MessLcd[6+Sens]);
}
break;

case 21: { // ----- Vitesse SPI
  EF_VitesseSPI=Sens;
  Lcd.print (MessLcd[29+Sens]);
}
break;

}
}

```

Test_Select.ino

```

// =====
// = Tests exécutés sur appui touche Select
// =====

void Test_Select() {

  Lcd.setCursor(0,1);

  switch (OrdreMenu[IdxMenu]) {

    case 00: { // ----- Insertion
      if (EF_AlimOk==1) {Arret_EF();}
      else {

```

```

        Init_EF();
        if (EF_InitOk==1) {Lcd.print(MessLcd[4]);}
      }
    }
    break;

    case 10: { // ----- Iris a "PO
      if (InitCmd()==0) {break;}
      Cmd_13PO();
    }
    break;

    case 12: { // ----- Raz Compteur AF
      if (InitCmd()==0) {break;}
      if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}

```

```

else {
    RazBuff_EF();
    BuffSendEF[1]=0x0A;
    BuffSendEF[3]=0x0C;
    BuffSendEF[5]=0xC0;
    SendBuff_EF(9);
    Tmp16a=(BuffRecEF[6].ValRet * 0x100) + BuffRecEF[7].ValRet;
    Lcd.setCursor(11,1);
    Lcd.print(Tmp16a,HEX);
    Lcd.print(" ");
}
}

```

```

break;
case 20: { // ----- IS On
    CptStab=millis();

}
break;

}
}

```

Test_Solo.ino

```

// =====
// = Tests exécutés une seule fois a chaque changement de menu
// =====

void Test_Solo() {

    Lcd.setCursor(0,1);

    switch (OrdreMenu[IdxMenu]) {

        case 00: { // -----Insertion
            Lcd.print (MessLcd[2+EF_AlimOk]);
        }
        break;

        case 01: { // -----Récupération paramètres
            if (InitCmd()==0) {break;}
            Cmd_80();
            if (EF_IdLens!=0) {
                for (Cpt=4; Cpt < 12; Cpt++) { LcdHex8(BuffRecEF[Cpt].ValRet); }
                Serial.println(NomMenu[OrdreMenu[IdxMenu]]);
                Serial.print(" Lens Id number = ");
                Serial.println(EF_IdLens);
                Serial.print(" Focale min = ");
                Serial.print(EF_Fmin);
                Serial.println ("mm");
                Serial.print(" Focale max = ");

```

```

Serial.print(EF_Fmax);
Serial.println ("mm");
Serial.print(" Fonction1 = ");
PrintHexBin8(EF_Fct1);
Serial.print(" Fonction2 = ");
PrintHexBin8(EF_Fct2);
if ((EF_Fct2 & 0x80) == 0x80) {
    Serial.print(" Extension ");
    for (Cpt=12; Cpt < 16; Cpt++) { PrintHex8(BuffRecEF[Cpt].ValRet); }
    Serial.println();
}
Serial.println();
}
}
break;

case 02: { // -----Récupération nom objectif
    if (InitCmd()==0) {break;}
    if ((EF_Fct2 & 0x80) == 0x80) {
        RazBuff_EF();
        BuffSendEF[1]=0x0A;
        BuffSendEF[3]=0x82;
        for (Cpt=4; Cpt < 32; Cpt++) {BuffSendEF[Cpt]=0x83;}
        SendBuff_EF(32);
        if (BuffRecEF[2].ValRet==0xAA && BuffRecEF[3].ValRet==0x00) {
            if (BuffRecEF[4].ValRet==0x82) {Lcd.print(MessLcd[12]);}
        }
        else {
            for (Cpt=4; Cpt<20 || BuffRecEF[Cpt].ValRet==0; Cpt++) {Lcd.write(BuffRecEF[Cpt].ValRet);}

```

```

Serial.print(NomMenu[OrdreMenu[IdxMenu]]);
Serial.print(" : ");
for (Cpt=4; Cpt<32 || BuffRecEF[Cpt].ValRet==0; Cpt++) {Serial.write(BuffRecEF[Cpt].ValRet);}
Serial.println();
}
}
else { ErreurCmd(); }
}
else {Lcd.print(MessLcd[13]);}
}
break;

```

```

case 10: { // ----- Iris a "PO
if (InitCmd()==0) {break;}
Cmd_13PO();
}
break;

```

```

case 12: { // ----- Aff Compteur AF
if (InitCmd()==0) {break;}
if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}
else {
Lcd.print(MessLcd[15]);
if ((Cmd_90() & 0x10) == 0x10) {
Lcd.setCursor(9,1);
Lcd.print("/");}
RazBuff_EF();
BuffSendEF[1]=0x0A;
BuffSendEF[3]=0xC0;
SendBuff_EF(7);
Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
Lcd.setCursor(11,1);
Lcd.print(Tmp16a,HEX);
}
}
break;

```

```

case 13: { // ----- Aff d map
if (InitCmd()==0) {break;}
if ((Cmd_90() & 0x80) == 0x80) { Lcd.print(MessLcd[14]);}
else {
Lcd.print(MessLcd[15]);

```

```

if ((Cmd_90() & 0x10) == 0x10) {
Lcd.setCursor(9,1);
Lcd.print("/");}
RazBuff_EF();
BuffSendEF[1]=0x0A;
BuffSendEF[3]=0xC2;
SendBuff_EF(7);
Tmp16a=(BuffRecEF[4].ValRet * 0x100) + BuffRecEF[5].ValRet;
Lcd.setCursor(10,1);
if (Tmp16a > 9999) {Lcd.print(" inf ");}
else {
Lcd.print(Tmp16a);
Lcd.print("cm"); }
}
}
break;

```

```

case 14: { // -----Vitesse AF
AffVitAF();
}
break;

```

```

case 15: { // ----- Eclairage objectif
Tmp8a=digitalRead(Led_Lens);
Lcd.print (MessLcd[8+Tmp8a]);
}
break;

```

```

case 16: { // ----- Debug On/Off
Lcd.print (MessLcd[6+FlagDebug]);
}
break;

```

```

case 21: { // ----- Vitesse SPI
Lcd.print (MessLcd[29+EF_VitesseSPI]);
}
break;

```

```

}
}

```

SPIxEos.h

```
struct SPIEOS_Ret
{
    uint8_t ValRet;
    uint16_t tRep;
};
```

SPIxEos.ino

```
//=== Envoi et réception 8bits en mode SPI Motorola_3 + contrôle temps réponse objectif Canon EOS ===
//=
//= Version minimale sans fonction retard, Temps Horloge0 = 170ns, H1= 500ns, Horloge 1.4Mhz env
//=
//= Octet envoyé = Valeur passé a la fonction au format int8_t
//= Retour fonction dans variable au format Structure SPIEOS_Ret : Octet reçu dans .ValRet, durée signal =
// de validation objectif dans .tRep. ou 0xFFFF si pas de signal, ou 0xFFFF si dépassement de durée
//=
// =====
```

```
// ----- Version transmission basse vitesse protocole EOS de base
// t Horloge0=5.9µs , tH1=6.5µs, Horloge= 80.3kHz
```

```
SPIEOS_Ret SPISend (uint8_t DataSend)
```

```
{
    uint8_t CptBit;
    uint8_t ValPort;
    uint8_t Retmp=0;
    SPIEOS_Ret ValRet;
```

```
for (CptBit=0; CptBit < 8; CptBit ++)
```

```
{
    ValPort=(DataSend & 0x80) ^ 0xFF; // Horloge a 1, DCL positionné = /Data
    PORTA=ValPort;
    DataSend = DataSend <<1;
    Retmp=Retmp <<1;
```

```
delayMicroseconds(7);
```

```
PORTA=ValPort & 0xBF; // Horloge a 0, DCL positionné = /Data
Retmp=Retmp | (PINA & 0x01);
delayMicroseconds(7);
}
```

```
delayMicroseconds(2); //Remise a zéro Data et Horloge
PORTA=0x3F;
ValRet.ValRet=Retmp;
ValRet.tRep=SPIEOS_tValid();
return(ValRet);
}
```

```
// ----- Version transmission haute vitesse
// t Horloge0=0.916µs , tH1=1.2µs, Horloge= 480kHz
```

```
SPIEOS_Ret SPIHSend (uint8_t DataSend)
```

```
{
    uint8_t CptBit;
    uint8_t ValPort;
    uint8_t Retmp=0;
    SPIEOS_Ret ValRet;
    volatile uint8_t DummyH=0; //Pour instructions bidon d'ajustement temps
```

```
noInterrupts();
for (CptBit=0; CptBit < 8; CptBit ++)
```

```

ValPort=(DataSend & 0x80) ^ 0xFF; // Horloge a 1, DCL positionné
PORTA=ValPort;
DataSend = DataSend <<1;
Retmp=Retmp <<1;
DummyH=PINA;
DummyH=DummyH >> 2;

PORTA=ValPort & 0xBF; // Horloge a 0, DCL positionné
Retmp=Retmp | (PINA & 0x01);
DummyH=DummyH >> 1;
}
interrupts();

delayMicroseconds(2); //Remise a 0 Data et Horloge
PORTA=0x3F;
ValRet.ValRet=Retmp;
ValRet.tRep=SPIEOS_tValid();
return(ValRet);
}

```

```

// -----Calcul durée signal validation-occupation niveau bas horloge objectif
// Retour = Durée signal, 0xFFFF si pas de validation, 0xFFFF si dépassement durée.

```

```

uint16_t SPIEOS_tValid()
{
    uint16_t Cpt;
    uint8_t Test;

    Test=(PINA & 0x02);
    Cpt=0x0000;

```

```

while (Test==2) { //Attente passage horloge au niveau bas
    Test=(PINA & 0x02);
    Cpt++;
    if (Cpt > 0x027) { //Si pas de reponse avant 30ms => valeur de retour = 0xFFFFE
        Cpt=0xFFFFE; //Valeur limite : 0x16=16µs, 0x20=28µs, 0x27=33µs,
        Test=00;
    }
}

if (Cpt < 0xFF00) { //Attente passage horloge au niveau haut
    Cpt=0x0000;
    Test=(PINA & 0x02);
    while (Test==0){
        Cpt++;
        if (Cpt > 0xFFFF0) { //Si tps low superieur a limite => valeur de retour = 0xFFFF
            Cpt=0xFFFF;
            Test=02;
        }
        delayMicroseconds(3); //Pour compteur =FFF0 t(1)=62ms, t(2)=107ms, t(3)=173.3ms (1 unité=2.8µs env.),
        t(4)=239ms
    }
}
return (Cpt);
}

```

Liens et révisions document

Sources et squetchs : http://jp79dsfr.free.fr/_Docs%20et%20infos/Sources/Protocole%20EF/

Protocole EF :

http://jp79dsfr.free.fr/_Docs%20et%20infos/Photo%20Tech%20_%20Canon%20EOS-EF%20Protocol.pdf

v1.00	16/10/2016	Première diffusion.
v1.10	25/10/2016	Ajout adaptateur version II