

IOCards

Dans cette page vous trouverez une brève vue d'ensemble de la technologie de matériel et logiciel IOCards d'OpenCockpits.

Cette technologie est bon marché, fiable, avérée, évolutive et leur logiciel est très puissant et flexible (et gratuit) !

Matériel : Je fournis quelques informations au sujet de l'utilisation des encodeurs rotatifs et sur les boîtes de break-out.

Logiciel : Une section importante détaille SIOC. Je donne des exemples de comment programmer avec toutes sortes de commutateurs et encodeurs dans SIOC (lié aux offsets de FSCONV) et nous décrivons également trois fonctions reliées qui ont enrichi mon expérience : support de cockpit cold & dark; essai de lumières (pour vérifier les défauts de fonctionnement de matériel) ; varier l'intensité des affichages (aussi bien en dimmer qu'en montant graduellement de foncé à lumineux).

vue d'ensemble

IOCards est un produit d'Opencockpits, un groupe - de constructeurs espagnols non commerciaux qui sont disposés à partager leur grande connaissance.

Une des choses que j'aime chez d'IOCards est que vous pouvez acheter la construction de cartes déjà montées et testées à bas prix.

J'ai deux cartes d'extension d'USB, une pour mon Overhead/MIP et une pour dans mon pedestal, 4 cartes master (deux reliées à chaque carte d'exp d'USB), 5 cartes de Display-II, plusieurs circuits imprimés d'afficheurs de 3 et 5 chiffres, et de nombreuses boîtes break-out pour un câblage facile.

Une carte master commande jusqu'à 45 sorties (pour des LED) et 72 entrées (pour des boutons, des commutateurs et des encodeurs rotatifs).

La possibilité de relier un encodeur rotatif directement à la carte master est un grand avantage !

Quatre cartes de Display-II peuvent être reliées à une carte master, chacune commandant jusqu'à 16 affichages de 7 segments (chiffres).

Il y a deux systèmes logiciels disponibles pour le matériel d'IOCards.

La version de base s'appelle IOCards et la version avancée s'appelle le SIOC.

Ce dernier fournit des possibilités très puissantes de programmation.

Voici l'écran de la dernière version (du 28 jan. 2007) de SIOC (3.4).

Cette version fournit le support de multiples cartes d'extension d'USB.



Dans la fenêtre de « dispositifs » (devices) vous pouvez identifier mes deux cartes d'extension d'USB. À la page de téléchargement vous trouverez le programme de SIOC (LevelD767_v4.xy.txt) pour mon cockpit basé sur le Level-D.

Du fait le programme vous trouverez des VARIABLEs avec l'attribut DEVICE 1, si elles sont liées à la carte master du pedestal.

Si aucun attribut n'est donné, le défaut (DEVICE 0) est utilisé.

SIOC

Puisqu'il y a beaucoup de confusion au sujet de la façon employer SIOC j'essayerai d'expliquer les « fondements » ici.

J'ai également compilé une présentation powerpoint sur une utilisation simple de SIOC (pour débiter)

J'en conviens, la documentation d'Opencockpits est énorme et débordante.

Mais pour débiter il vaut mieux ne pas utiliser plusieurs PCs, et laisser de côté le protocole d'IOCP et les « serveurs »...

On a besoin seulement d'un éditeur de texte comme le bloc-notes pour éditer une configuration, du programme config_sioc.exe pour compiler votre configuration dans un dossier exécutable (avec extension .ssi) et le programme sioc.exe lui-même naturellement pour l'exécuter.

Des paramètres pour SIOC sont (une seule fois) définis dans le fichier sioc.ini.

En outre le programme controlador.exe est très maniable pour vérifier si votre matériel d'IOCards fonctionne correctement ou pas.

Je donnerai deux exemples, le premier est un qui fonctionnera avec presque n'importe quel avion dans FS9 parce qu'il emploie un offset générique de FS2004 - FSUIPC, et le second est propre à Level-D parce qu'il fonctionne avec un offset de Level-D - FSUIPC (mais ces exemples intéressent tout le monde, je pense).

Il est très important de comprendre que SIOC n'est pas un langage de programmation exécutable normal avec un début et une fin, mais un langage activé par des événements. Qu'est ce que cela veut dire ? Que SIOC fonctionne avec des définitions de variables.

Les variables représentent des liens au monde extérieur, tel que des offsets de FSUIPC, entrées d'IOcards, des sorties ou des affichages d'locards, ou représentent des états ou des sous-programmes internes du programme.

Si une variable change de valeur ALORS (et seulement alors) le code de la variable contenu dans son corps, entre les accolades {}, est exécuté.

Ceci peut faire changer la valeur d'une autre variable, et ainsi vous pouvez obtenir une série d'événements.

C'est idéal pour simuler des logiques complexes d'un avion !

Exemple 1 : Interrupteur de phares d'atterrissage.

Cet exemple est basé sur l'offset FSUIPC générique pour les phares, il marchera avec presque n'importe quel avion.

Voici le code de SIOC :

```
Var 1, Link IOCARD_SW, Input 50, Type I
{
  v2 = CHANGEBIT 2, v1
}
```

```
Var 2, Link FSUIPC_OUT, Offset $0D0C, Length 2
```

Explication : La variable 1 est liée à votre interrupteur à levier "ON/OFF", relié à l'entrée 50 de la carte principale d'IOCards. (Notez que 50 est juste un exemple).

« Type I » désigne un commutateur "Marche/Arrêt".

Si le commutateur est allumé, la variable 1 vaut 1 et si le commutateur est éteint, la variable 1 vaut 0. La variable 2 est liée à l'offset FS2004 générique FSUIPC pour les commandes de phares (longueur de 2 bytes).

Le bit 2 dans cet offset représente les phares d'atterrissage (voir FSUIPC for Programmers.doc). Chaque fois que vous basculez votre commutateur, le corps (la partie entre les accolades) de la variable 1 est exécuté.

Il y a seulement un énoncé dans le corps ; il signifie que le bit 2 de la variable 2 sera fixé selon la valeur de la variable 1.

Ainsi si l'interrupteur à levier est sur ON, le bit 2 vaudra 1, et si le commutateur est sur OFF le bit 2 vaudra 0.

Notez qu'avec la fonction « Changebit » aucun autre bit de l'offset 0x0D0D ne sera modifié.

Exemple 2 : Commutateur principal des annunciators Caution et Warning de Level-D.

Pour ceci nous avons besoin de deux sorties vers des LEDs et l'entrée d'un bouton-poussoir.

Voici le code de SIOC :

```
Var 4100, Link FSUIPC_IN, Offset $8BBD, Length 1
{
  v4180 = TESTBIT v4100, 0
  v4181 = TESTBIT v4100, 1
}
```

```
Var 4180, Link IOCARD_OUT, Output 21
Var 4181, Link IOCARD_OUT, Output 22
```

```
Var 4200, Link IOCARD_SW, Input 5, Type P
{
  v4299 = CHANGEBIT 0, v4200
}
```

```
Var 4299, Link FSUIPC_OUT, Offset $8B16, Length 1
```

Explication :

Dans cet exemple vous voyez cinq définitions de variables (4100, 4180, 4181, 4200 et 4299). Il doit leur être donné un numéro qui leur est propre.

J'emploie les nombres qui correspondent à la section dans FSCONV où l'information des voyants d'avertissement et d'alarme est définie, mais vous pouvez employer les nombres que vous souhaitez. Les variables peuvent également avoir un nom facultatif joint. Je le fais toujours, cela augmente la lisibilité de la source, mais dans cet exemple je l'ai omis.

Voyons d'abord les LEDs :

La variable 4100 est un lien d'entrée avec l'offset FSUIPC 0x8BBD de 1 byte.

Si ce byte change de valeur, le code dans le corps de la variable 4100 sera exécuté.

La fonction de TESTBIT donnera à la variable 4180 la valeur 1 si le byte 0 de la variable 4100 vaut 1.

Ceci vaut pour la variable 4181 qui prendra la valeur de 1 si le byte 1 de cette variable vaut 1.

Autrement les valeurs de la variable 4180 et de la variable 4181 seront 0.

Ceci aura comme conséquence un changement de la valeur de la variable 4180 ou de la variable 4181.

Ces Variables sont liées à deux sorties d'IOCard (21 et 22) et de cette façon nous contrôlons nos LEDs.

Ainsi si par exemple la variable 4180 change de valeur, cela sera transmis à la sortie 21 et donc la lumière s'éteindra ou s'allumera.

Puis les commutateurs (plus au sujet des commutateurs dans SIOC ici) :

La variable 4200 est liée à un commutateur par l'entrée 5 de l'IOCard.

« Type P » signifie qu'IOcards gardera son état interne, ainsi si vous poussez il passera à 1, si vous le poussez encore passera à 0, et ainsi de suite.

Chaque fois que vous poussez, l'état interne change de valeur et le code de corps de la variable 4200 est exécuté ; et la fonction CHANGEBIT règlera le bit 0 de la variable 4299.

Noter que les autres bits de la variable 4299 ne seront pas affectés.

La variable 4299 est liée en sortie à l'offset FSUIPC 0x8B16 de 1 byte.

Si vous examinez la documentation de FSCONV vous verrez que le bit 0 de ce byte commandes le commutateur principal (Master Switch) de level-D.

C'est tout !

Fondamentalement très simple, et facile pour à implémenter avec plus de logique.

Par exemple supprimer l'activation d'une lumière en état « cold and dark », ou en ajoutant une fonction de test des voyants.

Les commutateurs dans SIOC

Cette section donne des exemples de la façon programmer dans SIOC avec des commutateurs matériels reliés à la carte master d'IOCards. Le rapport avec les offsets de FSCONV est également donné.

Il y a plusieurs types de commutateurs matériels.

Je décrirai ici :

- le bouton poussoir (SPST) ;
- l'interrupteur "Marche/Arrêt" avec deux bornes (SPST) ;
- le commutateur rotatif (avec n bornes, où $n \geq 2$) ;
- le commutateur "Marche/Arrêt" avec trois bornes (SPDT) et l'encodeur rotatif gris.

Pour plus d'informations sur les commutateurs, voir sur Wikipedia

le bouton poussoir (SPST)

Est un commutateur qui ferme un circuit électrique quand il est poussé et ouvre le circuit quand il est relâché (non poussé).

Il a deux bornes, une doit être reliée à la masse d'un groupe d'entrées de la carte master et à l'autre à une des 9 entrées de ce même groupe.

Quelle borne est reliée à quoi n'a pas d'importance.

Si le bouton n'est pas poussé l'entrée est haute (1) et si poussé l'entrée est basse (0).

Dans SIOC on code un bouton comme ceci :

```
Var 1, name Button, Link IOCARD_SW, Input 23, Type P
```

«Type P » signifie que SIOC verrouillera l'état du bouton, ainsi chaque fois que le bouton est poussé, l'état de la variable 1 basculera (de 0->1, de 1->0, et ainsi de suite).

Si le bouton est poussé l'état change et le code attaché à la variable est exécuté.

Voici un exemple de mon code de SIOC du FMC, le bouton Next Page :

```
Var 6226, name NEXT_PAGE, Link IOCARD_SW, Input 23, Type P
{
  &FO_FMCFunc = CHANGEBIT 26, &NEXT_PAGE
}
```

Chaque fois que le bouton est enfoncé, le bit 26 dans la variable de sortie FO_FMCFunc change de valeur. Ceci aura comme conséquence que FSCONV envoie la commande de Next_Page au Level-D.

Notez que les fonctions dans FSCONV qui peut être mises en application de cette façon sont indiquées avec T (pour Toggle) dans la colonne « type » de la table d'offsets de commande.

Une autre manière possible d'utiliser un bouton dans SIOC est (noter le « Type I ») :

```
Var 11, name LightsTestButton, Link IOCARD_SW, Input 103, Type I
{
  &LightsTest = &LightstTestButton
}
```

La variable 11 vaudra 1 si le bouton n'est pas poussé et 0 si le bouton est poussé. Donc il n'y a aucun verrouillage de l'état comme avec le type P

Une autre manière d'utiliser un bouton dans SIOC est :

```
Var 12, Link IOCARD_SW, Input 23, Type P
{
  &Flag = CHANGEBITN 0, &Flag
}
```

Dans cet exemple la valeur de la Variable 12 n'importe pas maintenant et n'est pas donc nommée ; chaque fois que le bouton est poussé le Flag changera de valeur.

Interrupteur "Marche/Arrêt" avec deux bornes (SPST) :

Ce commutateur a deux positions mécaniques différentes.
 En une position le contact est fermé et en l'autre position le contact est ouvert.
 Le type le plus populaire est celui à bascule avec un petit levier et ceux à bouton.
 Les bornes doivent être reliées à la carte master comme un bouton.
 Dans SIOC on code un commutateur "ON/OFF" avec deux bornes comme ceci :

```
Var 2, name OnOff2, Link IOCARD_SW, Input 23, Type I
```

```
// I signifie "on/off" switch
```

La variable 2 vaut 0 ou 1 selon la position physique du commutateur (à levier ou à poussoir, peu importe)

Voici un exemple de mon code de MCP de SIOC, pour le commutateur d'automanette (AT) :

```
Var 4601, name ATSwitch, Link IOCARD_SW, Input 70, Type I
{
  &FO_Switches = CHANGEBIT 1, &ATSwitch
}
```

Le Bit 1 de la variable de sortie FO_Switches (qui sera lue par FSCONV) reflétera exactement la position du commutateur, et ainsi FSCONV enverra une commande à Level-D chaque fois que cet état changera.

Note : Les fonctions dans FSCONV qui peut être mis en application de cette façon sont indiquées par un O dans la colonne Type de la table d'offsets de commande.

On peut également mettre en application des fonctions d'OL (L=Leading) de FSCONV de cette façon. La seule différence est le comportement de FSCONV. FSCONV va non seulement à chaque changement d'état envoyer une commande au Level-d mais il vérifiera sans interruption si le switch dans le simulateur est en même position que celui du cockpit ; et sinon il mettra le simulateur dans le même état que le cockpit en envoyant une commande au Level-d.

Commutateur rotatif (n positions, n >=2)

ce commutateur a n positions mécaniques différentes.
 Une borne est à la masse commune et les autres n bornes représentent chaque position du commutateur rotatif.
 La masse commune doit être mise à la masse d'un groupe de 9 entrées d'une carte master et les n autres bornes doivent être reliées à n entrées du même groupe.

Dans SIOC on code un commutateur rotatif avec n positions (dison 3) par n variables, une pour chaque entrée de la carte principale :

Var 31, name RotPos1, Link IOCARD_SW, Input 23, Type I // I means "on/off" switch.

Var 32, name RotPos2, Link IOCARD_SW, Input 24, Type I

Var 33, name RotPos3, Link IOCARD_SW, Input 25, Type I

Selon la conception mécanique du commutateur rotatif seulement un contact peut être fermé à la fois, ainsi seulement une des trois Variables vaudra 1, et les autres zéro, quand le commutateur est dans une position.

Voici un exemple de mon code SIOC « Passenger Sign », le commutateur NoSmoking :

```
Var 3800, name NoSmokOff, Link IOCARD_SW, Input 12, Type I
```

```
{
  &FO_PassSig = CHANGEBIT 0, &NoSmokOff
}
```

```
Var 3801, name NoSmokAut, Link IOCARD_SW, Input 13, Type I
```

```
{
  &FO_PassSig = CHANGEBIT 1, &NoSmokAut
}
```

```
Var 3802, name NoSmokOn, Link IOCARD_SW, Input 14, Type I
```

```
{
  &FO_PassSig = CHANGEBIT 2, &NoSmokOn
}
```

Les bits 0,1 et 2 dans la variable de sortie FO_PassSig (lu par FSCONV) refléteront toujours la position du commutateur, ainsi FSCONV enverra la bonne commande au Level-D quand la position change.

Comme pour le commutateur "ON/OFF", il y a deux sémantiques possibles.

La sémantique de type rotatif (indiqué avec R dans FSCONV) pour lequel une commande est générée seulement lors d'un changement d'état, et le type RL (Rotary Leading) qui envoie non seulement une commande au changement d'état mais aussi lorsque FSCONV détecte une différence entre le commutateur dans le simulateur et le commutateur matériel.

Interrupteur "ON/OFF" avec trois bornes (SPDT)

Les On/of avec deux bornes posent un petit problème dans SIOC.

Au démarrage la position de début ne peut pas être déterminée de manière 100% fiable.

Elle dépend de la position du commutateur et la manière dont il est relié.

Ainsi ce n'est pas fiable et peut nécessiter une correction manuelle après, ce qui est légèrement ennuyant, particulièrement si vous avez un cockpit avec beaucoup de commutateurs.

C'est pourquoi voici la version "ON/OFF" à 3 bornes (SPDT single pole double throw).

Ce commutateur a deux positions mécaniquement différentes.

C'est un commutateur appelé « change over », il y a toujours une position ouverte et une fermée.

Les plus populaires sont le type à levier et le type à bouton.

Une borne est la masse commune et les deux autres bornes sont pour chaque position.

La masse commune doit être mise à la masse d'un groupe de 9 entrées d'une carte master et les deux autres bornes doivent être reliées à deux entrées dans le même groupe.

Dans SIOC on code un commutateur "ON/OFF" à 3 bornes avec deux Variables, une pour chaque entrée de la carte master :

Var 40, name OnOff0, Link IOCARD_SW, Input 23, Type I // I signifie "on/off" switch.

Var 41, name OnOff1, Link IOCARD_SW, Input 24, Type I

La Variable 40 vaut 0 ou 1 selon la position physique du commutateur (levier ou bouton peu importe).
La variable 41 vaut 0 ou 1 selon la position physique du commutateur et les Variables 40 et 41 ont toujours des valeurs différentes!

Voici un exemple de mon code de MCP de SIOC, le commutateur de FlightDirector du pilote :

```
Var 4602, name CapFD0, Link IOCARD_SW, Input 110, Type I
{
  &FO_Switches = CHANGEBIT 4, &CapFD0
}
```

```
Var 4603, name CapFD1, Link IOCARD_SW, Input 111, Type I
{
  &FO_Switches = CHANGEBIT 5, &CapFD1
}
```

Les bits 4 et 5 de la variable de sortie (lue par FSCONV) refléteront la position du commutateur, ainsi FSCONV pourra envoyer une commande au Level-d chaque fois que le commutateur change de position.

Comme pour le commutateur à deux bornes, il y a deux versions.

Le type rotatif (indiqué R dans FSCONV) pour lequel une commande est générée seulement au changement d'état, et le type de RL (Rotary Leading) qui envoie non seulement une commande au changement d'état mais aussi si FSCONV détecte une différence entre le simulateur et le commutateur matériel.

Noter que le comportement d'un commutateur de SPDT est exactement identique à celui d'un commutateur rotatif à 2 positions.

L'avantage de cette solution sur la version de SPST est qu'au démarrage la position initiale du commutateur est toujours détectée correctement par SIOC et ainsi 100% fiable.

L'inconvénient est l'utilisation d'une entrée supplémentaire de la carte master.

Que se passe-t-il si FSCONV ne supporte seulement qu'un commutateur SPDT ("ON/OFF" avec 3 bornes) avec 2 bits mais que vous voulez employer un commutateur SPST ("ON/OFF" avec deux bornes) et juste 1 entrée de carte master ?

Aucun problème, vous pouvez relier votre commutateur juste comme n'importe quel autre SPST.

Revenons à l'exemple du Flight Director du pilote du MCP.

Avec juste une entrée disponible le voici :

```
Var 4602, name CapFD, Link IOCARD_SW, Input 7, Type I
{
  &FO_Switches = CHANGEBIT 4, &CapFD
  &FO_Switches = CHANGEBITN 5, &CapFD
}
```

Ainsi à chaque poussée nous produisons les deux bits dont FSCONV a besoin.

De cette façon n'est pas présente naturellement la fiabilité au démarrage des trois bornes, mais cela fonctionne dès que le simulateur et le matériel seront synchronisés.

Encodeur rotatif gris

L'encodeur rotatif a trois bornes : une masse commune et une pour chaque direction.

On relie un encodeur rotatif gris directement à la carte master comme un commutateur "ON/OFF" à 3 bornes (voir ci-dessus), à la restriction que vous devez employer deux entrées logiques consécutives de la carte principale.

Dans SIOC on code un encodeur rotatif comme ceci (noter le type) :

```
Var 5, name RO_HDG, Link IOCARD_ENCODER, Input 40, Aceleration 2, Type 2
```

L'entrée 40 est la première entrée logique, la deuxième entrée est entrée 41.

Voici un exemple de mon code de MCP de SIOC, le rotatif HDG :

```
Var 4641, name RO_HDG, Link IOCARD_ENCODER, Input 40, Aceleration 2, Type
2
{
  L0 = &RO_HDG      // * -1 turning right should be plus
  &HDG = ROTATE 0, 359, L0
}
```

La variable HDG contiendra la valeur du cap variant de 0 à 359.

Noter que la direction dépend de la manière dont vous avez relié les bornes aux entrées, ainsi si la rotation à droite n'incrémente pas les valeurs (mais les diminue) vous pouvez changer cela tout à fait facilement dans SIOC (sans changements de matériel) par l'intermédiaire de la multiplication par -1, ainsi le corps contiendrait alors :

```
{L0 = &RO_HDG * -1 // turning right should be plus
&HDG = ROTATE 0, 359, L0
}
```

L'état « cold and dark »

Quand le pilote et les co-pilotes montent dans le Boeing 767-300 pour la première fois le matin, l'habitacle est « cold and dark ».

Le panneau de Boeing 767-300 Level-D simule ceci parfaitement en ne montrant aucune lumière ou chiffre et en ne produisant aucun bruit.

Cependant, sans ajouter de modofocaitons spéciales, certaines des LED et affichages à 7 segments dans votre home cockpit seront allumés.

Ce n'est pas ce que vous voudriez.

C'est pourquoi j'ai ajouté un offset à FSCONV qui indique l'état du panneau.

Je montrerai comment j'emploie cet offset pour créer la même sensation de « cold and dark » dans votre cockpit.

Ceci est réalisé dans le logiciel SIOC d'Opencockpits.

D'abord je crée une variable qui lit l'état du cockpit dans le bit 0 d'un offset à un byte de FSUIPC à l'adresse 0x8BC5.

Cet offset est produit par FSCONV, vous pouvez le trouver dans la section 9.1

```
Var 9100 name FI_Panel Link FSUIPC_IN Offset $8BC5 Length 1
{
  &ColdAndDark = TESTBIT &FI_Panel 0
}
```

L'état est copié dans une autre variable appelée ColdAnddark :

```
Var 300 name ColdAndDark
{
  CALL &Refresh
}
```

Chaque fois que l'état change, le sous-programme Refresh est appelé.

Ce que fait ce sous-programme sera décrit plus loin.

D'abord voici une règle importante de conception de mon code de cockpit de SIOC :

Chaque sortie, LED ou affichage à 7 segments, n'est pas directement associé à une valeur, mais est commandé par l'intermédiaire d'un sous-programme appelé « de sortie ».

Vous pouvez toujours reconnaître une telle variable parce que son nom commence par « out ».

Prenons par exemple les annonceurs de Warning & Caution dans le commutateur principal (section 4.1 de FSCONV).

```
Var 4100 name FI_WarCau Link FSUIPC_IN Offset $8BBD Length 1
{
  CALL &OutWarCau
}
```

```
Var 4120 name OutWarCau Link SUBROUTINE
{
  IF &ColdAndDark = 1
  {
    &O_Warning = 0
    &O_Caution = 0
  }
  ELSE
  {
    &O_Warning = TESTBIT &FI_WarCau 0
  }
}
```

```

    &O_Caution = TESTBIT &FI_WarCau 1
  }
}

```

```

Var 4180 name O_Warning Link IOCARD_OUT Output 92
Var 4181 name O_Caution Link IOCARD_OUT Output 93

```

La variable 4100 montre que le sous-programme OutWarCau est appelé quand l'offset 0x8BBB change de valeur.

C'est la situation normale quand l'état de l'un de ces deux annonceurs a changé ou tous les deux. La variable 4120, le sous-programme OutWarCau, montre qu'il n'assigne pas simplement aux sorties 92 et 93 la valeur des nouveaux états d'annonceurs Warning & Caution, mais elle teste D'ABORD (!) la valeur de la variable ColdAndDark.

Si ces dernières valent 1 alors les LED seront éteintes indépendamment de ce que peuvent être les valeurs des panneaux.

Donc revenons finalement au sous-programme Refresh, il ressemble à ceci :

```

Var 302 name Refresh Link SUBROUTINE
{
  CALL &OutWarCau
  CALL &OutVOR1Crs // see further down ...
  // and here the CALL's to all other Out-routines controlling leds and 7-segment
  displays
}

```

Le sous-programme Refresh appelle aussi la fonction OutWarCau, juste comme c'est fait dans la situation normale par la variable 4100, quand un annonceur a changé d'état.

Ainsi chaque fois que la variable ColdAndDark change la valeur (de deux manières), Refresh s'appellera, avec l'effet que quand vous commencez en cold and dark les LEDs seront éteintes et quand vous enclenchez le commutateur de batterie, l'état de ColdAnddark est 0, la routine Refresh aura l'effet que les bonnes LEDs seront allumées.

La même approche est également adoptée pour commander les affichages à 7 segments ; par exemple voyez les codes de VOR1 Course :

```

Var 4321 name OutVOR1Crs Link SUBROUTINE
{
  IF &ColdAndDark = 1
  {
    &D_VOR1Crs = -999999 // blank display
  }
  ELSE
  {
    &D_VOR1Crs = &VOR1Crs // to display
  }
}

```

```

Var 4371 name D_VOR1Crs Link IOCARD_DISPLAY Digit 0 Numbers 3

```

Ainsi si l'habitacle est « cold and dark », les affichages seront éteints, autrement ils montreront le Vor 1 Course actuel.

En résumé : en combinant les informations fournies par FSCONV sur l'état cold and dark des panneaux du Level-D et une approche de programmation dans SIOC nous pouvons facilement rendre réelle l'expérience fascinante d'un vrai habitacle cold and dark !

Noter que tous mes exemples de SIOC dans les sources du SIOC pour Level-D à la page de téléchargements prennent en compte la fonction cold and dark.

Tests d'affichage

Les vrais cockpits ont des boutons pour des tests d'affichages.

Je crois que pour le 767 vous devez enfoncer un tel bouton pendant 3 secondes pour activer l'essai et encore pousser pour stopper le test.

Ces essais sont pratiques pour examiner le matériel et déceler les défauts de fonctionnement.

C'est également vrai dans nos cockpits-maison, particulièrement s'ils se développent avec un bon nombre de LEDs, de fils, et d'affichages, ainsi vous avez un test facile pour voir si tout travaille correctement.

Certains essayent de réaliser ceci avec du matériel spécial et des circuits, j'ai opté pour une solution intégralement logicielle.

J'ai mis ceci en application avec l'approche de programmation suivante :

Un bouton-poussoir est relié à la variable 230.

Étant donné que la variable 230 est le type I ceci aura comme conséquence que la variable 230 vaudra 1 quand le bouton est poussé et 0 quand le bouton n'est pas poussé (légèrement différent des vrais boutons de 7676, mais pas trop compliqué pour l'exemple).

```
Var 230 name LtsTest Link IOCARD_SW Input 103 Type I
{
  CALL &Refresh
}
```

Chaque fois que je pousse ou je libère le bouton la routine Refresh s'appellera, (pour la routine Refresh voir « cold and dark »).

Les sous-programmes de sortie pour des LED et affichages à 7 segments ont un test supplémentaire comparé à l'exemple du « cold and dark ».

Après le test du cold and dark, ainsi on ne peut pas réaliser un test dans une situation cold and dark (comme en vrai), ils vérifient que la touche "TEST" est poussée ; si oui les LEDs seront à 1, sinon le sous-programme de sortie placera les LED aux valeurs réelles.

Notez que parce Refresh s'appelle également quand le bouton-poussoir est libéré, la situation avant le test de lumières sera rétablie (après le test)

```
Var 4120 name OutWarCau Link SUBROUTINE
{
  IF &ColdAndDark = 1
  {
    &O_Warning = 0
    &O_Caution = 0
  }
  ELSE
  {
    IF &LtsTest = 1
    {
      &O_Warning = 1
      &O_Caution = 1
    }
    ELSE
    {
      &O_Warning = TESTBIT &FI_WarCau 0
      &O_Caution = TESTBIT &FI_WarCau 1
    }
  }
}
```

```
}  
}
```

La routine de VOR1 Course avec le test cold and dark et des lampes ressemble maintenant à ceci :

```
Var 4321 name OutVOR1Crs Link SUBROUTINE  
{  
  IF &ColdAndDark = 1  
  {  
    &D_VOR1Crs = -999999 // show a blank display  
  }  
  ELSE  
  {  
    IF &LtsTest = 1  
    {  
      &D_VOR1Crs = 888 // show three digits '8'  
    }  
    ELSE  
    {  
      &D_VOR1Crs = &VOR1Crs // show crs value at display  
    }  
  }  
}
```

Notez que tous mes exemples de SIOC dans les archives de sources SIOC pour Level-D de la page téléchargements prennent en compte les test de lampes.

Affichages variant en intensité

Les affichages à 7 segments d'une carte de Display-II d'Opencockpits peuvent être variés en intensité (versions depuis mars 2006).

C'est un bon dispositif parce que vous pouvez ajuster l'éclat selon les conditions d'environnement. On varie en envoyant une valeur au chiffre 15 de la carte de Display-II (de 0..15). L'inconvénient est que le chiffre 15 ne peut plus être employé pour l'affichage.

L'exemple de code de la façon dont vous pouvez varier un affichage peut être trouvé dans mes archives de sources SIOC à la page de téléchargement, de la série de HOW_TO....

Je décrirai ici une autre application du variateur.

Imaginez que nous sommes dans l'état cold and dark.

Quand vous poussez le commutateur de batterie, les chiffres des affichages, tels que le Vor, jauges de carburant ... montent graduellement !

Comment faisons-nous cela ?

D'abord nous avons placé une variable au chiffre 15 de la carte de Display-II :

```
Var 70 name D_Displ Link IOCARD_DISPLAY Digit 15 Numbers 1
```

Alors nous définissons un sous-programme de gradation qui envoie des valeurs à ce chiffre 15 de la carte d'affichage :

```
Var 5 name Dimmer Link SUBROUTINE
```

```
{
  &D_Displ = -999994
  &D_Displ = &Dimmer
}
```

Alors nous définissons la variable qui réalise la gradation par l'intermédiaire d'un temporisateur, et appelle la fonction Refresh afin de montrer les valeurs quand elles augmentent : (pour plus d'information au sujet de la fonction Refresh voir la section « cold and dark »)

```
Var 1 name DispBoot // boot displays up after Cold and Dark, they gradually come to Life!
```

```
{
  L0 = &DispBoot
  CALL &Dimmer L0
  IF L0 = 0
  {
    &DispBoot = TIMER 15 1 25 // from 0 to 15 in steps of 1, each step takes 0.25
    seconds
  }
  ELSE
  {
    IF L0 = 1
    {
      CALL &Refresh
    }
  }
}
```

C'est un code un peu compliqué parce que cette variable s'assigne périodiquement elle-même, svp lisez la soigneusement (ou copiez la seulement...)

Enfin nous avons laissé ce processus commencer dans la variable ColdAndDark, que nous avons déjà vue dans les autres exemples.

Afin de faire la gradation j'ai changé le corps de cette variable.

Quand nous allons d'un état cold and dark à un état normal il assignera 0 à la variable de DispBoot qui déclenche la gradation du processus.

```
Var 300 name ColdAndDark
{
  IF &ColdAndDark = 0
  {
    &DispBoot = 0 // let DispBoot call Refresh during boot up after it has cleared the
display
  }
  ELSE
  {
    CALL &Refresh
  }
}
```

Notez que tout le code normal pour les valeurs d'affichage, comme la variable 4321 OutVor1CRS dans l'exemple de test, n'est pas effectuée du tout.

Une bonne solution.

Les encodeurs rotatifs

Notez la différence entre un commutateur rotatif (1 x n) et un encodeur rotatif...

la carte master d'IOCards offre trois options différentes en ce qui concerne les encodeurs rotatifs.

D'abord vous pouvez fabriquer un prétendu encodeur simulé à partir d'un commutateur 1x12 rotatif en combinant les bornes en trois groupes de quatre (alternativement, ainsi 1.4.7.10 et 2.5.8.11 et 3.6.9.12), et les relier à trois entrées consécutives de la carte master de sorte que dans la rotation

DANS LE SENS DES AIGUILLES D'UNE MONTRE les sorties aux trois groupes de broches sont :

100 - 010 - 001 - 100...

et en anti-horlogique :

100 - 001 - 010 - 100...

le logiciel d'IOCards (logiciel d'encodeur de commutateur) est capable de détecter ce modèle et la direction de rotation.

Dans le logiciel de configuration pour encodeurs vous pouvez indiquer l'incrément et le facteur d'accélération, à appliquer en tournant le bouton rapidement.

2) Une seconde et une possibilité très intéressante est que le logiciel d'IOCards peut manipuler les encodeurs gris directement s'ils sont reliés à deux entrées (logiques) consécutives de la carte master ! Le code gris est celui ou la sortie change seulement d'un bit à la fois.

Par exemple, prenons un code binaire à deux bits.

Quand le compte va vers le haut de l'ordre de code est :

00 (zéro), 01 (un), 10 (deux), 11 (trois), 00 (zéro)

Notez qu'en allant de un à deux à trois puis à zéro les deux bits vont d'1 à 0.

Ainsi, plus que un bit unique a changé simultanément en passant de 3 à 0.

Maintenant, voici le même ordre, mais dans le code gris :

00 (zéro), 01 (un), 11 (deux), 10 (trois), 00 (zéro).

Notez qu'en allant de un à deux à trois puis à zéro seulement un bit change à la fois.

Donc le type gris signifie, **DANS LE SENS DES AIGUILLES D'UNE MONTRE** :

00-01-11-10-00

et anti-horlogique :

00-10-11-01-00

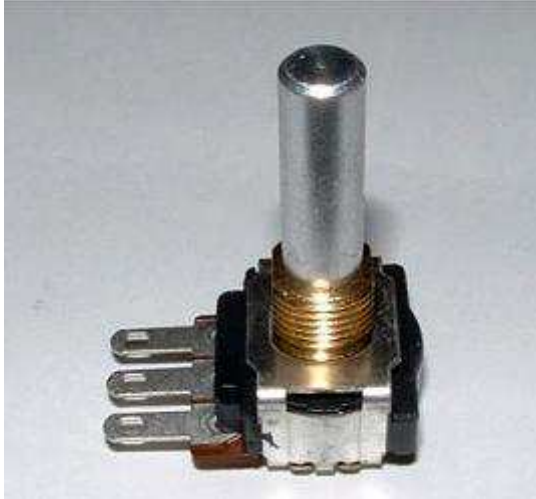
Une condition préalable supplémentaire pour l'usage avec IOcards est que l'encodeur change seulement d'1/4 de cycle par détente !

(notez qu'il y a également un type d'encodeurs gris qui changent d'1 cycle par détente ou d'1/2 cycle par détente)

IMPORTANT : il vous faut des ENCODEURS GRIS TYPE 2 BITS AVEC 1/4 CYCLE PAR DÉTENTE :

Exemple : REWP ou RENP, disponible chez Flightdeck Solutions, page « matériel » (réellement de chez CTS, type 288)

Un encodeur qui est de type gris et fonctionnera est ce type d'encodeur rotatif (RENP, de type 288 de chez CTS) chez Flightdeck Solutions (Peter Cos) :



Des exemples de la façon de programmer un encodeur gris dans SIOC peuvent être trouvés [ici](#).

3) La troisième option est d'employer une carte Rotary Encoders-II :



Ce circuit peut lire les données de 4 encodeurs différents.

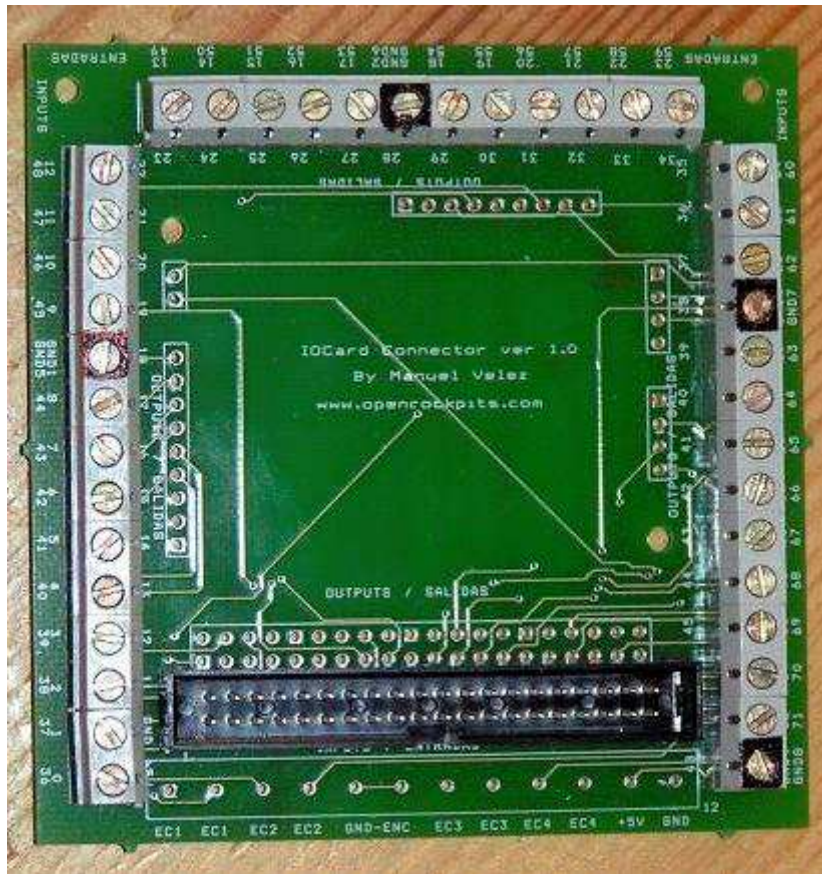
Je ne recommande pas cette carte puisqu'elle complique votre matériel, est plus chère, et pas vraiment nécessaire (seulement si vous voulez employer les encodeurs optiques). Il y a de bonnes solutions de rechange, comme décrit ci-dessus.

Les boîtes de break-out

Opencockpits de déblocage propose également des boîtes de break-out. Vous pouvez relier vos entrées et des sorties à la mastercard, sans devoir souder.

La boîte de BO (ou la carte de raccordements comme ils l'appellent) est Bi-sexuelle... Selon le placement du connecteur à 40 broches elle est employée en tant que carte à 4 * 10 (9 + la masse) ou 38 sorties avec une masse.

Voici la version entrée :



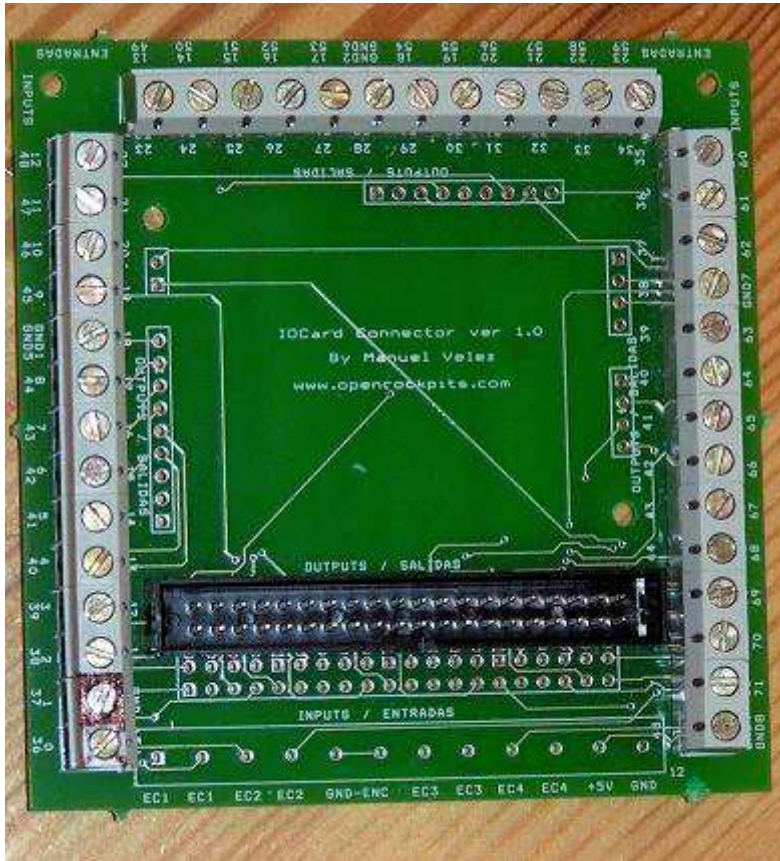
Les masses de chaque groupe sont indiquées dans cette image par une tache noire.

Les broches sont bien indiquées.

Notez que 9 bornes d'entrées partagent une même masse, la numérotation des broches indiquée est la numérotation logique.

Si vous reliez un encodeur gris vous devrez prendre les broches n et n+1 d'un même groupe.

Et voici légèrement plus difficile à interpréter la version de sorties :



Noter que la borne 1/37 est la MASSE (en 2^{ème} position à partir du côté inférieur gauche, voir la tache noire).

La borne 0/36 est +5V (pas vraiment requis) et les 38 autres bornes sont les sorties.