NOTE D'APPLICATION

Simulation modèle contrôleur PWM décrit en C++ avec PSpice.

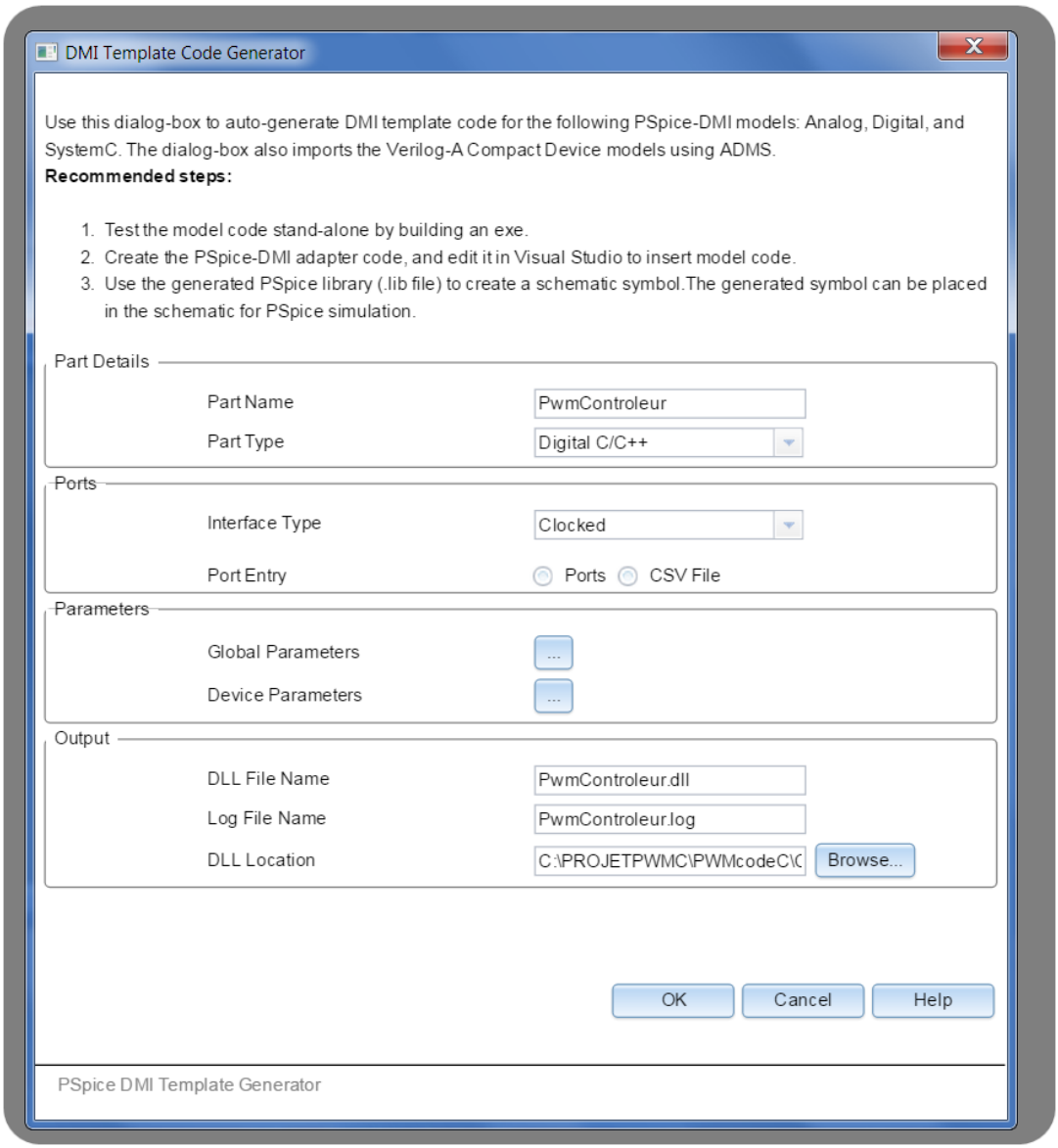
Cette note explique comment simuler un modèle de contrôleur PWM ou de MLI (modulation de la largeur d'impulsion) décrit en langage C++ avec PSpice.
Ce modèle intégré dans une librairie de liens dynamiques (DLL) est créé à partir d'un projet Visual C++.
C'est à l'aide d'une nouvelle interface disponible dans PSpice Model Editor que ce projet est généré selon les entrées-sorties et les paramètres souhaités.
Tous les fichiers nécessaires à sa compilation y sont déclarés notamment ceux permettant une adaptation avec le cœur du simulateur PSpice.
En final, ce modèle est utilisé dans un convertisseur Buck DC-DC pour contrôler la commutation.

PREREQUIS

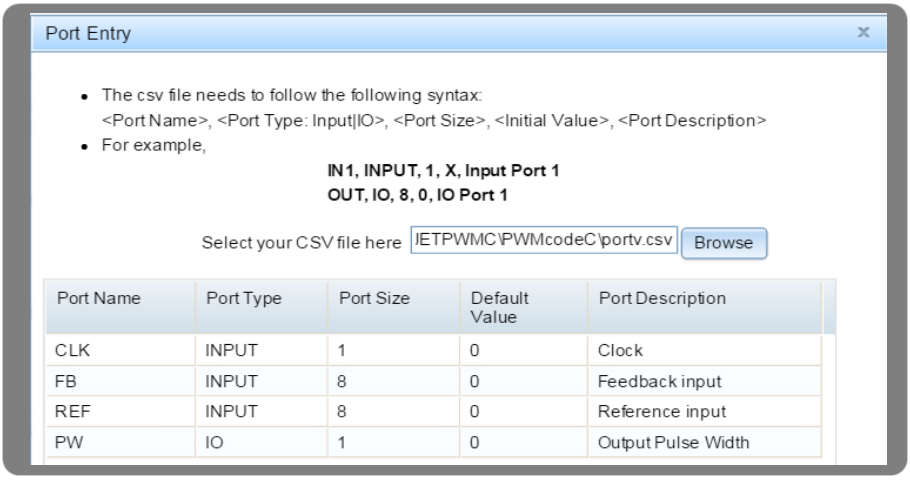
- **OrCAD PSpice Designer Plus ou AMS Simulator Version 17.2**
- **Microsoft Visual Studio Community 2013.**

CREATION DU PROJET VISUAL C++.

1. **Ouvrez le projet DCDCPWMC.opj dans CAPTURE. PS : Choisissez une licence PSpice Designer Plus ou AMS Simulator.**
2. **Ouvrez PSpice Model Editor via le menu File/New /PSpice Library.**
3. **Lancez l'interface pour créer le projet C++ : menu Model/DMI Template Code Generator.**

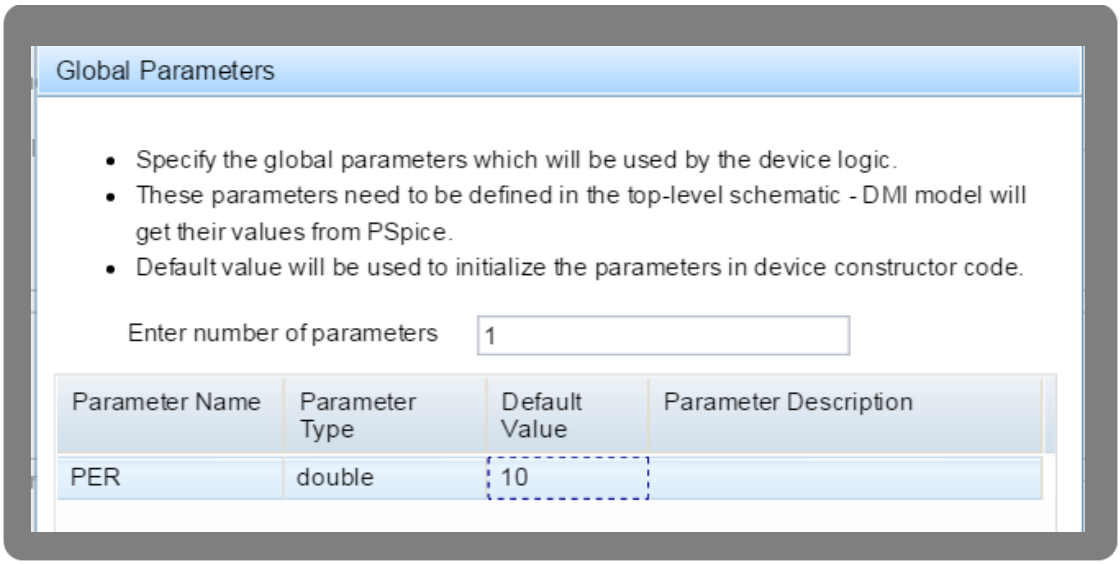


4. **Dans le champ Part Name, éditez le nom du modèle PwmControleur.**
5. **Sélectionnez Clocked dans le type d'interface des ports. Le modèle du contrôleur PWM est en effet séquentiel.**
6. **Dans la section Output, cliquez sur le bouton Browse pour choisir le répertoire du projet Visual C++. Nommez le répertoire CODE et placez-le sous celui du projet Capture.**
7. **Dans la section Ports, cliquez sur le bouton CSV File puis bouton Browse pour lire le fichier portv.csv (répertoire du projet) contenant la définition des ports.**



Les entrées logiques du contrôleur sont un bus REF de 8 bits, un bus FB de 8 bits et une horloge CLK. Le modèle décrit en code C++ commandera l'état logique de la sortie PW.

8. **Puis OK pour fermer la boîte de dialogue Port Entry.**
9. **Dans la section Parameter, cliquez sur le bouton Global Parameters.**
10. **Editez 1 dans le champ Enter number of parameters puis Apply.**
11. **Renseignez le nom « PER » et la valeur du paramètre par défaut « 10 » en double-cliquant dans les cases d'édition.**

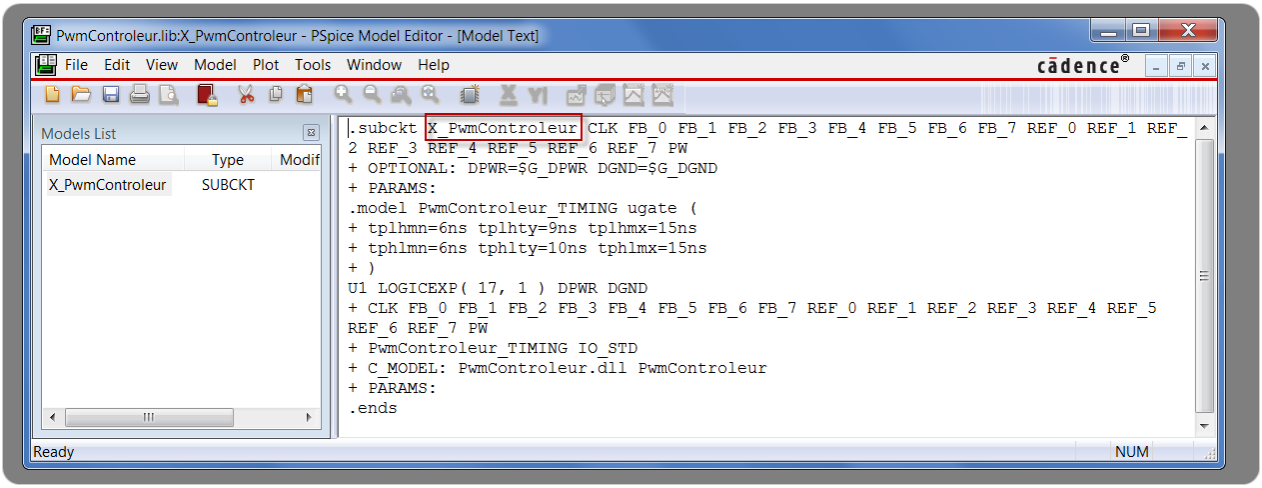


Le paramètre PER permet de définir la période du signal numérique modulé ; celle-ci sera égale à (PER*Période de l'horloge). Ce paramètre sera modifiable depuis le schéma.

12. **Puis OK pour fermer la boîte de dialogue Global Parameters.**
13. **Puis OK pour générer le projet Visual C++, les fichiers sources et le fichier modèle de simulation PwmControleur.lib.**

Un fichier rapport PwmControleur.log est également généré et s'affiche. Vérifiez-le puis fermez-le.

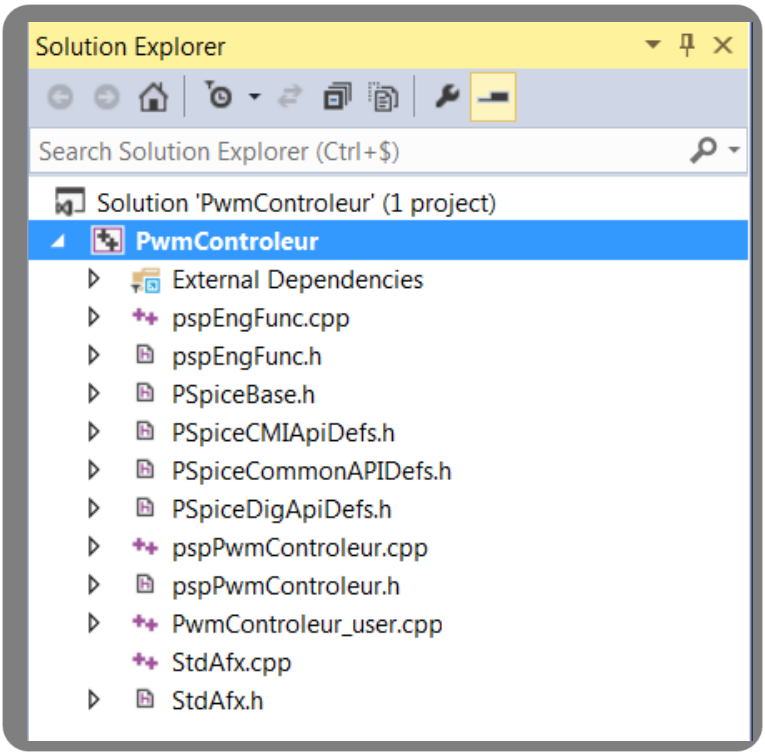
```
Changes in evaluate function...
Digital/SystemC GUID creation...
read input file CommonTemplate/pspEngFunc_model.cpp
opened output file C:/PROJETPWMC/PWMcodeC/CODE//pspEngFunc.cpp for writing
read input file CommonTemplate/pspEngFunc_model.h
opened output file C:/PROJETPWMC/PWMcodeC/CODE//pspEngFunc.h for writing
read input file DigitalTemplate/pspDigitalModelName.cpp
opened output file C:/PROJETPWMC/PWMcodeC/CODE//pspPwmControleur.cpp for writing
read input file DigitalTemplate/pspDigitalModelName.h
opened output file C:/PROJETPWMC/PWMcodeC/CODE//pspPwmControleur.h for writing
read input file DigitalTemplate/digitalmodel_user.cpp
opened output file C:/PROJETPWMC/PWMcodeC/CODE//PwmControleur_user.cpp for writing
Populating vcproj with the source and header files
read input file DigitalTemplate/DigitalModelName.vcxproj
Lib created..
```



15. Renommez le modèle PwmControleur.
16. Puis File/Save.
17. Puis File/Exit pour quitter PSpice Model Editor.

CREATION ET COMPILATION DU CODE.

1. Lancez Visual Studio 2013.
2. Menu File/Open/Project pour ouvrir le fichier PWMControleur.vxproj du répertoire CODE du projet.
3. Dans la fenêtre Solution Explorer, cliquez sur l'icône flèche devant PwmControleur pour visualiser l'ensemble des fichiers ajoutés au projet.



3. Double-cliquez sur le fichier PwmControleur_user.cpp pour l'ouvrir. C'est dans celui-ci que le code décrivant le modèle doit être écrit.

4. **Avec la barre de défilement, repérez les lignes de codes suivantes :**

```
REF[3]=pVectorStates[12].getLevel();
REF[4]=pVectorStates[13].getLevel();
REF[5]=pVectorStates[14].getLevel();
REF[6]=pVectorStates[15].getLevel();
REF[7]=pVectorStates[16].getLevel();
PW=pVectorStates[17].getLevel();


}
```

Ces instructions sont réalisées lorsqu'un front montant est détecté sur le signal horloge.
C'est entre la dernière instruction PW=pVectorStates[17].getLevel() et la fermeture de l'accolade } que le contrôleur PWM qui est séquentiel pourra être codé.

5. **Editez les lignes de code suivantes :**

```
PW = pVectorStates[17].getLevel();

pspBits2Int(FB, FBint, 8);
pspBits2Int(REF, REFint, 8);

if (REFint > FBint && mD < 0.98)
{
    mD += 0.01;
}
else if (REFint < FBint && mD > 0.02) {
    mD -= 0.01;
}

if (mCurrentCLKCount <= 0) { mCurrentCLKCount = mPER; }


if (mCurrentCLKCount > mD*mPER)
    mPWStatus = false;
else
    mPWStatus = true;

if (mPWStatus == true && (int)PW != 1)
{
    PW = pspBit::HI;
}
else if (mPWStatus == false && (int)PW != 0)
{
    PW = pspBit::LO;
}

mCurrentCLKCount--;

}
```

Dans un premier temps, la fonction pspBits2Int convertit les bus PSpice (FB, REF) en entier FBint et REFint.
Ces entiers sont ensuite comparés pour incrémenter ou décrémenter la valeur du paramètre mD, sa valeur est limitée entre 0.02 et 0.98.
Un entier mCurrentCLKCount est introduit pour décompter le nombre de front montant de l'horloge. Sa valeur est initialisée à mPER qui est la valeur du paramètre PER. La dernière instruction « mCurrentCLKCount-- ; » le décrémente de 1. Sa valeur varie donc cycliquement de PER à 1.
Cet entier est ensuite comparé à la quantité mD*mPER. Selon le résultat de cette comparaison le booléen PWStatus est vrai ou faux.
Enfin le signal PW passe de 0 à 1 si le booléen PWStatus est vrai et passe de 1 à 0 si le booléen est faux.

<u>Explications :</u>
Lorsque FBInt est inférieur à REFint, le paramètre mD (0 au départ) est incrémenté de 0.01 à chaque front montant d'horloge. Au bout d'un certain temps, la quantité mD*mPER est supérieure à 1 (valeur minimale de mCurrentCLKCount), le signal PW passe à 1 et dans le même temps, la valeur mCurrentCLKCount est décrémentée de 1 donc passe à 0.
Au front d'horloge suivant, mD est incrémenté de 0.01 et mCurrentCLKCount est réinitialisé à sa valeur maximale mPER. La quantité mD*mPER devient alors inférieure à mCurrentCLKCount : le signal PW passe à 0. La durée minimale d'une impulsion est donc celle d'une période d'horloge.
Si FBint est toujours inférieur à REFint, la quantité mD*mPER est ensuite supérieure à 2. Le signal PW passe à 1 et le reste pendant 2 périodes d'horloge. Et ainsi de suite…
Au bout d'un certain temps, mD arrive à sa limite maximale c'est-à-dire 0.98. La quantité 0.98*mPER est inférieure à mCurrentCLKCount pour une seule valeur de mCurrentCLKCount : sa valeur maximale mPER. Le signal PW passe donc à 0 pendant une seule période d'horloge.
Lorsque FBInt est inférieur à REFint, la durée de l'impulsion varie donc d'une période d'horloge à (mPER-1) périodes d'horloge.
Si on suppose que FBint devient alors supérieur à REFint, la durée de l'impulsion va diminuer progressivement. La valeur minimale de mD est 0.02. La quantité mD*mPER peut être supérieur à 1 si mPER est supérieur à 50.
Dans cet exemple, mPER est égale à 10. La quantité mD*mPER minimale est donc 0.2 : celle-ci est toujours inférieure à mCurrentCLKCount (varie de 1 à 10). A partir d'un certain temps, le signal PW reste donc à 0.

6. **Modifiez la ligne de code suivante, comme décrit ci-dessous:**

```
int j = 17;
for (int i = 0; i <1; i++) {
//if (oldPW == PW) continue;
lState = (pVectorStates)[j];
lState = PW;
fp_SetState(mRef, j, &lState, NULL);
j++;
}
```

```
int j = 17;
for (int i = 0; i <1; i++) {
//if (oldPW == PW) continue;
lState = (pVectorStates)[j];
lState = PW;
fp_SetState(mRef, j-17, &lState, NULL);
j++;
}
```

7. **File/Save PwmControleur_user.cpp.**
8. **Ouvrez le fichier pspPwmControleur.h depuis la fenêtre Solution Explorer.**
9. **Ajoutez le code suivant pour déclarer l'entier mCurrentCLKCount, le réel mD et le booléen PWStatus. Notez que les entiers FBint et REFint ont été automatiquement déclarés lors la création du projet.**

```
unsigned int CLKint,FBint,REFint,PWint;
pspBit  FB[8], REF[8], CLK, PW, oldPW;
int mCurrentCLKCount;
bool mPWStatus;
double mD;
```

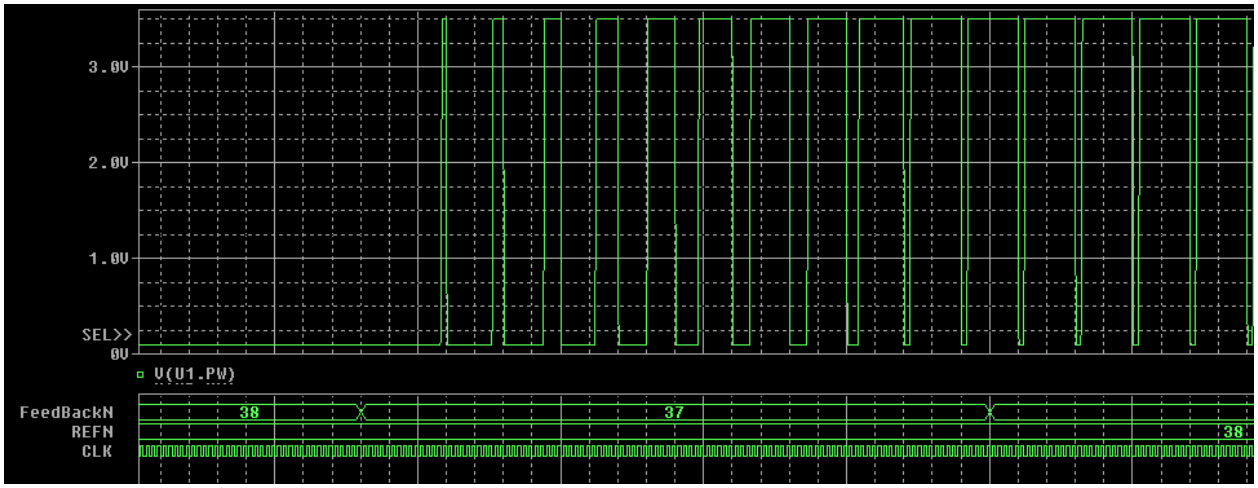
10. **File/Save pspPwmControleur.h.**
11. **Menu Build/Configuration Manager.**
12. **Choisissez Release comme Configuration et X64 comme Platform. Puis Close.**
13. **Menu Build/Build Solution pour créer la DLL qui intègre le modèle PWMControleur. Le fichier PwmControleur.dll est créé dans le répertoire CODE/x64/Release dans le dossier du projet.**

PS : Un projet de secours est présent dans le répertoire CODE_FINAL du projet en cas d'erreur non solutionnée lors de la compilation de votre code.

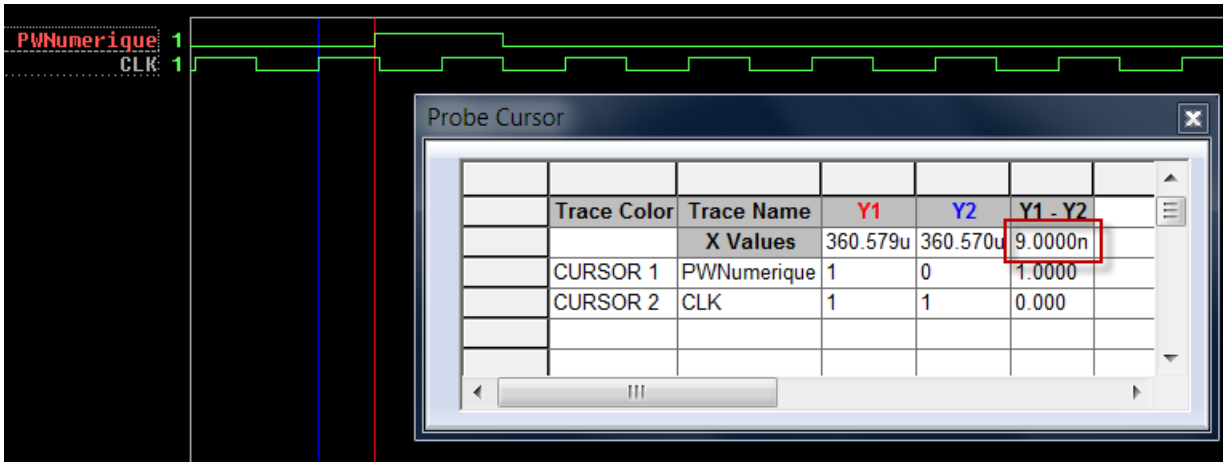
UTILISATION DU CONTROLEUR PWM.

1. **Avec l'explorateur Windows, copiez la DLL PwmControleur.dll dans le répertoire du profil de simulation DCDCPWMC-PSpiceFiles\ConvertisseurBuck\tran pour qu'elle soit lue par PSpice lors de la simulation.**
2. **Lancez la simulation.**
3. **Pour visualiser les résultats ci-dessous, accédez au menu Window/Display Control et double-cliquez sur la présentation souhaitée.**

Tant que la tension de sortie est inférieure à la référence (12V), le contrôleur génère des impulsions.



La largeur des impulsions varie d'une période d'horloge (20ns) à (PER-1) périodes soit 180ns.
La valeur de PER est définie dans le schéma ConvertisseurBuck, vous pouvez modifier sa valeur pour visualiser son impact sur les résultats.



Des délais ont été automatiquement ajoutés au modèle PwmControleur.
A l'aide des curseurs on mesure 9ns entre le front d'horloge et le passage de 0 à 1 du signal PW.
En éditant le modèle, ils sont facilement modifiables :

```
.subckt PwmControleur CLK FB_0 FB_1 FB_2 FB_
+ OPTIONAL: DPWR=$G_DPWR DGND=$G_DGND
+ PARAMS:
.model PwmControleur TIMING ugate (
+ tplhmn=6ns tplhty=9ns tplhmx=15ns
+ tphlmn=6ns tphlty=10ns tphlmx=15ns
+ )
U1 LOGICEXP( 17, 1 ) DPWR DGND
```

CONCLUSION : Via la nouvelle interface disponible dans PSpice Model Editor, la version 17.2 permet la création de modèles numériques de haut niveau décrit en langage C++. Ces modèles intégrés dans une librairie DLL permettront de simuler des systèmes complexes analogiques/numériques avec PSpice.

Artedas FRANCE

1 bis avenue Foch
94100 Saint Maur - France
Tél : +33(0)1 70 91 67 20
Fax: +33(0)1 70 91 67 29
E-mail: info@artedas.fr

www.artedas.fr