Synthèse de Circuits utilisant l'Arithmetique Virgule Fixe

Maminionja Ravoson

Roselyne Chotin-Avot / Thibault Hilaire

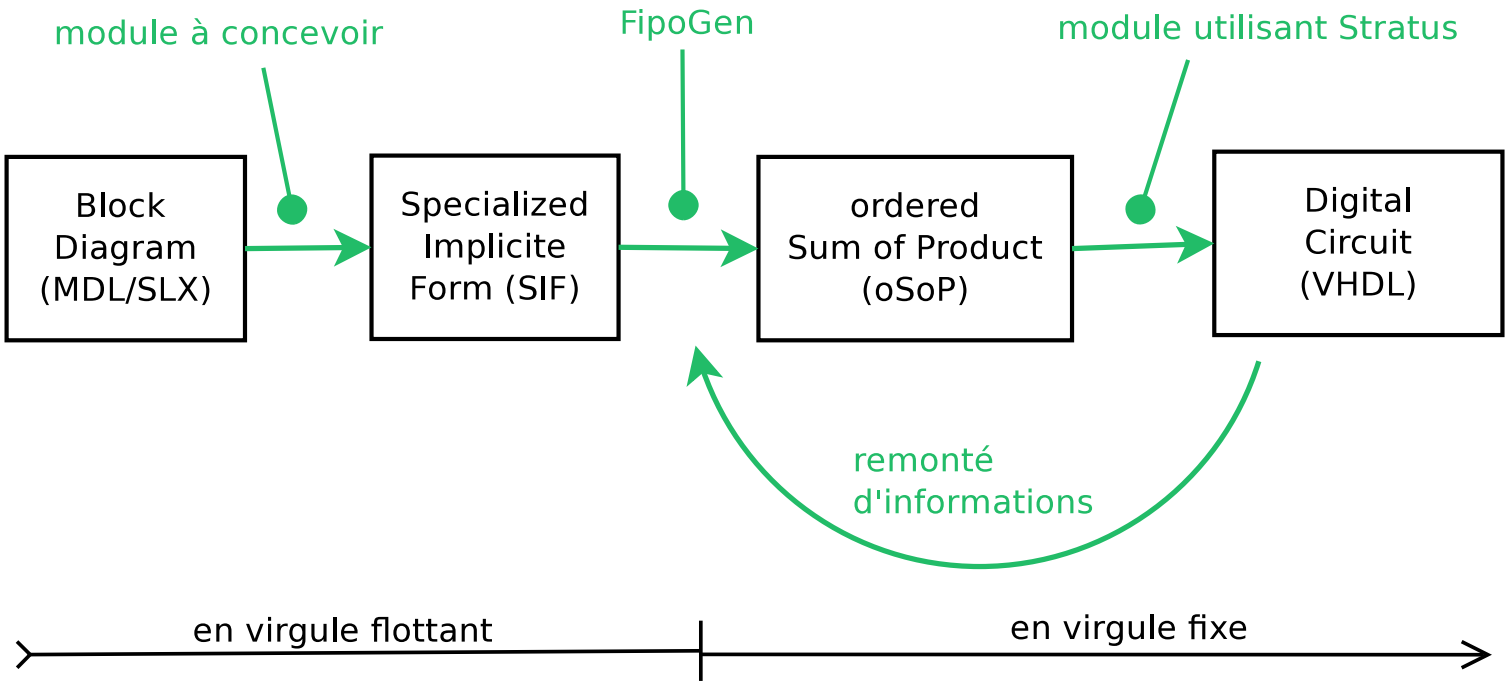
07 Juillet 2015

Pourquoi ce stage ?

- Le stage fait partie du projet **FxPSynthesis** - Action initiative du LIP6
  - Collaboration entre les 2 équipes CIAN et PEQUAN
  - Développement d'un **outil commun pour la conception de circuits**
  - Dans la thématique Systèmes embarqués
  - Un flot complet - accélerer le développement d'application
- Les participants:
  - **CIAN** (Circuits Intégés Analogiques et Numériques)
    Dpt Système sur Puce
  - **PEQUAN** (Performance et Qualité des Algorithmes Numériques)
    Dpt Calcul Scientifique

Le sujet

- **Objectif**: avoir un outil de synthèse de haut-niveau pour les filtres linéaires, en utilisant l'arithmétique virgule fixe



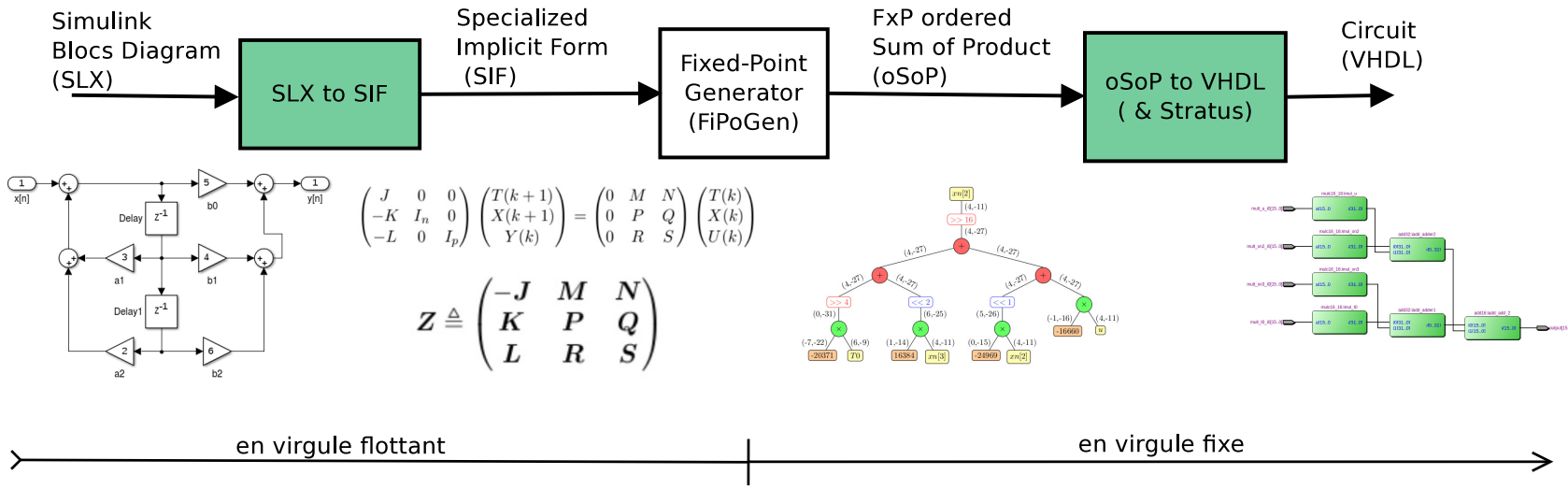
Le sujet

- **Etat de l'art**:
  - Des outils de synthèse de filtres existe déjà, ex: FDAtool de Matlab
  - Par simulations/raffinements - pas très performants.
- **Le projet** permet de développer un outil performant commun entre les 2 équipes.

  Á notre disposition, on a:
  - **FiPoGen**: un outil qui permet de transformer un algo en du code virgule fixe déjà spécifié
  - **Stratus**: qui permet de décrire des composants matériels à l'aide de ces générateurs paramétrables

Le sujet

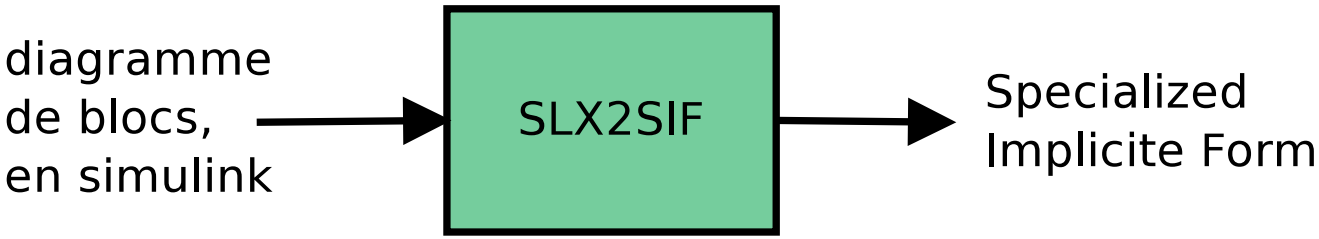
- Schéma de l'outil à développer



- **les tâches** spécifiques au stage:
  - Concevoir le module "SLX to SIF"
  - Développer le module "oSoP to VHDL" à l'aide de Stratus
  - Intégrer ces modules aux outils existant

Définition et analyse du problème

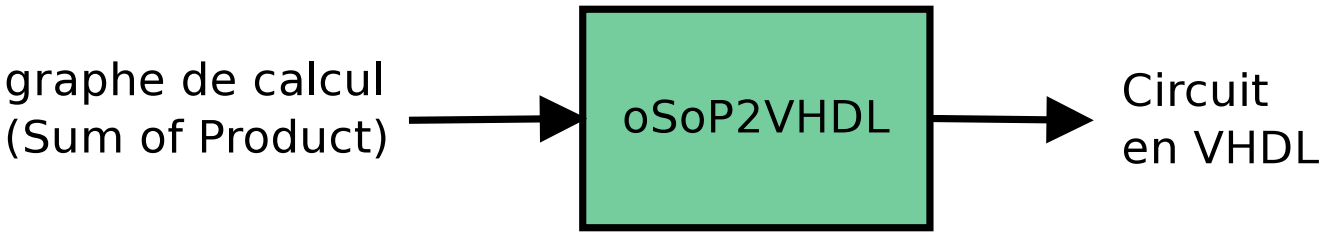
Le module d'entrée pour FiPoGen (SLX2SIF):



- Les fichiers de diagramme simulink ne sont pas bien documentés, donc il faut faire un peu de **reverse-engineering**
- Un minimum de compréhension sur **l'algèbre linéaire, la théorie des graphes et les filtres numériques**
- Comprendre la représentation en **S.I.F** des systèmes linéaires
- Trouver l'**algorithme** pour la transformation

Définition et analyse du problème

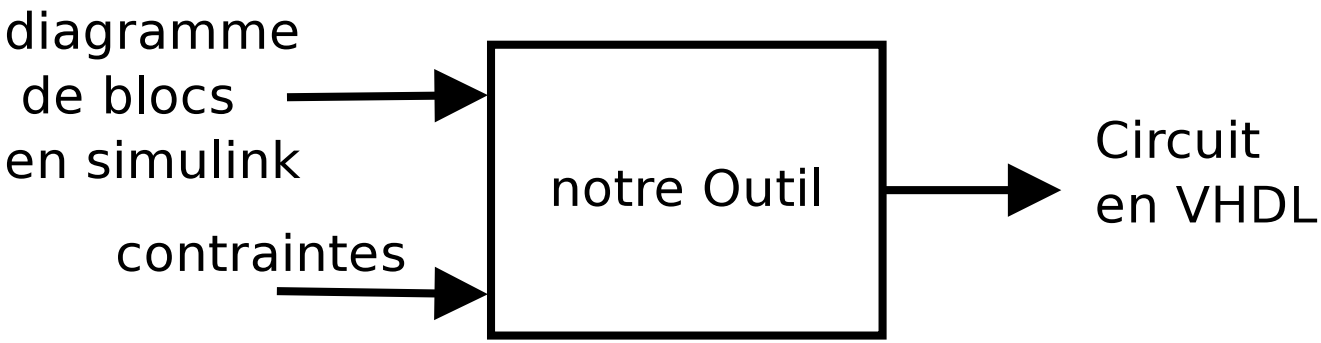
Le module de sortie avec Stratus (oSoP2VHDL):



- Avoir une notion sur l'**arithmétique virgule fixe**
- Comprendre un peu l'outil **FiPoGen**, en particulier la **classe oSoP**
- Savoir utiliser efficacement les **Générateurs de Stratus**
- Un mécanisme pour **valider** que le code vhdl généré correspond bien à l'oSoP

Définition et analyse du problème

L' **intégration** des modules developpés:
On veut avoir un flot de conception "automatique" fonctionnel

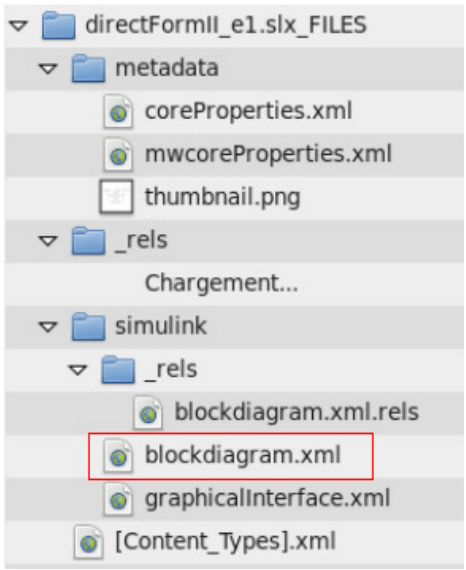


- Les modules doivent être **compatible** et forment une chaine uniforme avec l'existant, **en Python**
- Il sera nécessaire de développer des **"glue logic"** pour mettre ensemble les modules
- Un **code réutilisable et bien documenté** pour être utilisé par d'autre à la fin du stage.

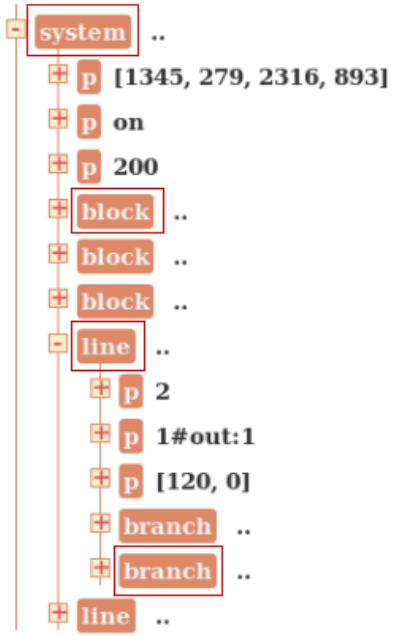
Principe de la solution

Pour le module **Simulink vers SIF** :

- SLX : conteneur de fichiers XML et non XML, conforme à la norme OPC(Open Packanging Convention)



Contenu d'un fichier SLX



L'élement <ModelInformation/Model/System> du fichier blockdiagram.xml

```
<Line>
    <P Name="ZOrder">19</P>
    <P Name="Src">13#out:1</P>
    <P Name="Points">[0, 20]</P>
    <Branch>
        <P Name="ZOrder">23</P>
        <P Name="Dst">11#in:1</P>
    </Branch>
    <Branch>
        <P Name="ZOrder">21</P>
        <P Name="Dst">14#in:1</P>
    </Branch>
</Line>
```

exemple de line avec 2 branches

```
<Block BlockType="Sum" Name="Sum2" SID="8">
    <P Name="Ports">[2, 1]</P>
    <P Name="Inputs">|++</P>
    <P Name="ZOrder">7</P>
    <P Name="InputSameDT">off</P>
    <P Name="IconShape">round</P>
```

exemple de bloc somme

Principe de la solution

- La forme implicite spécialisé (SIF) :

$$\begin{pmatrix} J & 0 & 0 \\ -K & I_n & 0 \\ -L & 0 & I_p \end{pmatrix} \begin{pmatrix} T(k+1) \\ X(k+1) \\ Y(k) \end{pmatrix} = \begin{pmatrix} 0 & M & N \\ 0 & P & Q \\ 0 & R & S \end{pmatrix} \begin{pmatrix} T(k) \\ X(k) \\ U(k) \end{pmatrix}$$

$$\begin{cases} \boldsymbol{Jt}(k+1) = & \boldsymbol{Mx}(k) + \boldsymbol{N}u(k) \\ \boldsymbol{x}(k+1) = \boldsymbol{Kt}(k+1) + \boldsymbol{Px}(k) + \boldsymbol{Q}u(k) \\ y(k) = \boldsymbol{Lt}(k+1) + \boldsymbol{Rx}(k) + \boldsymbol{S}u(k) \end{cases}$$

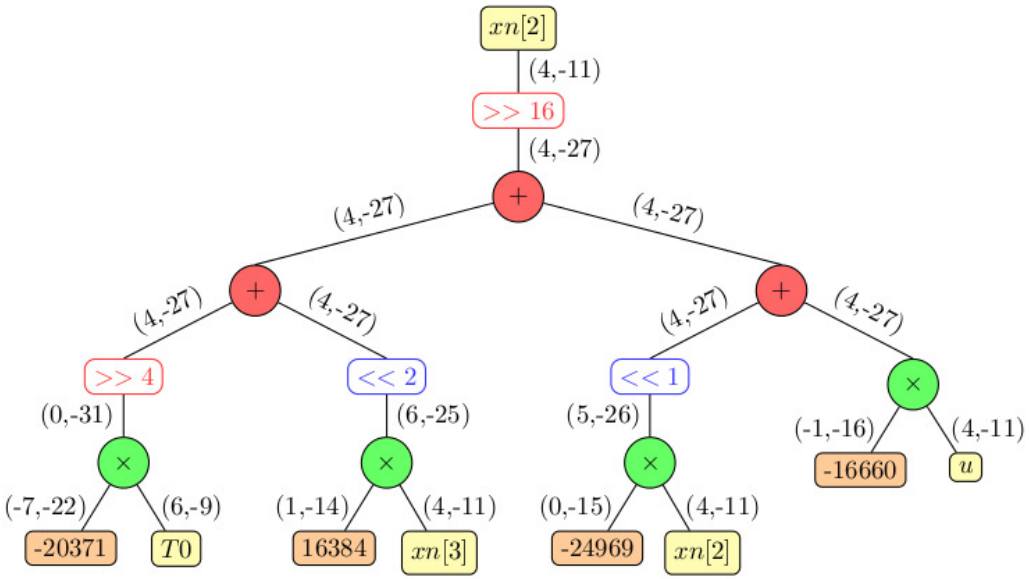
t(k+1) : résultat intermédiaire
y(k) : sortie
x(k+1) : état suivant
u(k) : entrée

$$\boldsymbol{Z} \triangleq \begin{pmatrix} -\boldsymbol{J} & \boldsymbol{M} & \boldsymbol{N} \\ \boldsymbol{K} & \boldsymbol{P} & \boldsymbol{Q} \\ \boldsymbol{L} & \boldsymbol{R} & \boldsymbol{S} \end{pmatrix}$$

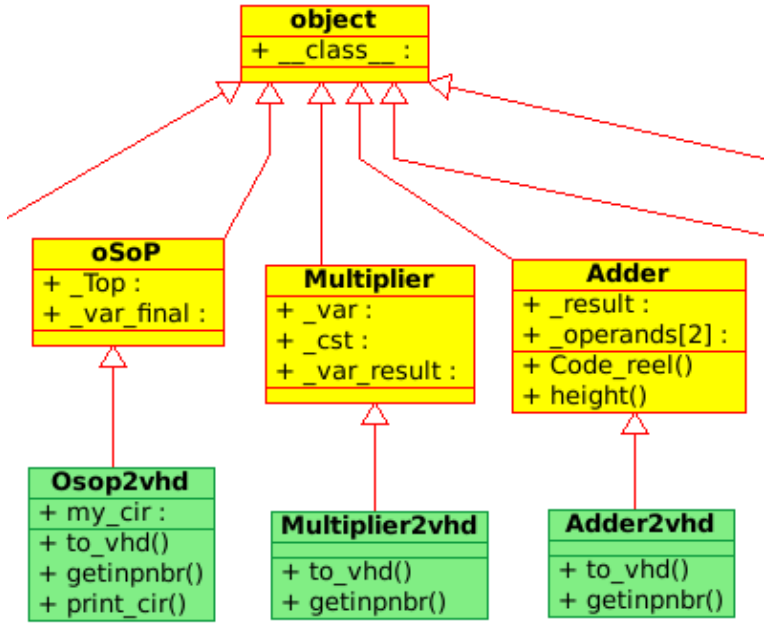
- Parcourir les blocs (t, x, y)
- Exprimer l'équation pour chaque bloc en f(t, x u)
- Aligner les équations et identifier les matrices
- Vérification : Faire à la main puis comparer

Principe de la solution

Pour le module **oSoP vers VHDL**



exemple d'oSoP



extrait diagramme de classe du module oSoP vers VHDL

- Étudier l'implémentation des oSoP dans FiPoGen
- Comprendre en détails les générateurs de Startus utiles pour notre cas
- Comprendre l'outil Stratus (le modèle objet associé) pour développer

Identification des tâches à accomplir

- **Prendre en main** le projet, le sujet du stage
- Maitriser l'utilisation des **outils**, Python en particulier
- Étudier la structure des **fichiers** matlab (SLX)
- Bien comprendre le **SIF** et les filtres numériques
- **Développement** SLX vers SIF
- Développement oSoP vers VHDL
- Test, **débogage** et optimisation
- **Intégration**
- **Rédaction** du rapport et des autres documentations

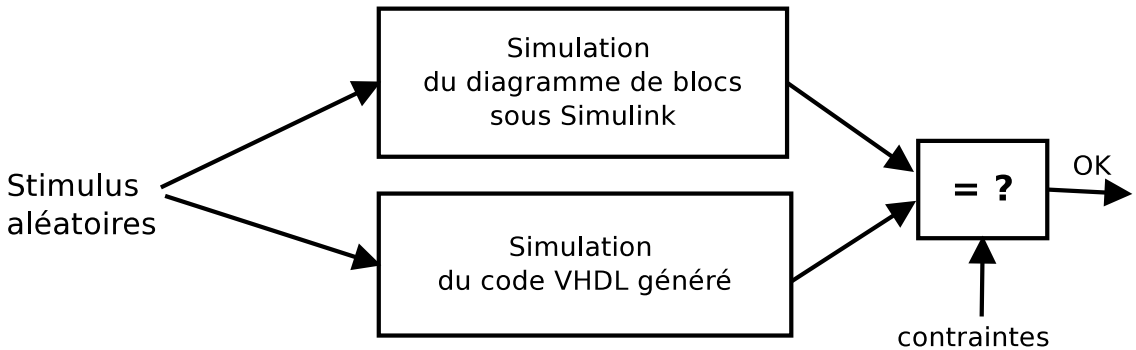
Identification des tâches à accomplir

Chaque **développement** est constitué par les étapes suivantes:

- Étude de l'existant
- Prototypage
- Test et validation
- Analyse des résultats et correction
- Intégration de la fonctionnalité

Procédure de recette

- Au niveau **développement** :
  utiliser des tests unitaires. Tous les tests devront passer
- Au niveau des **fonctionnalités** :
  On testera par des exemples déjà traités par les 2 équipes
- Au niveau de la **chaine complète** :



Procédure de recette

Pour la partie **diagramme Simulink vers SIF** :

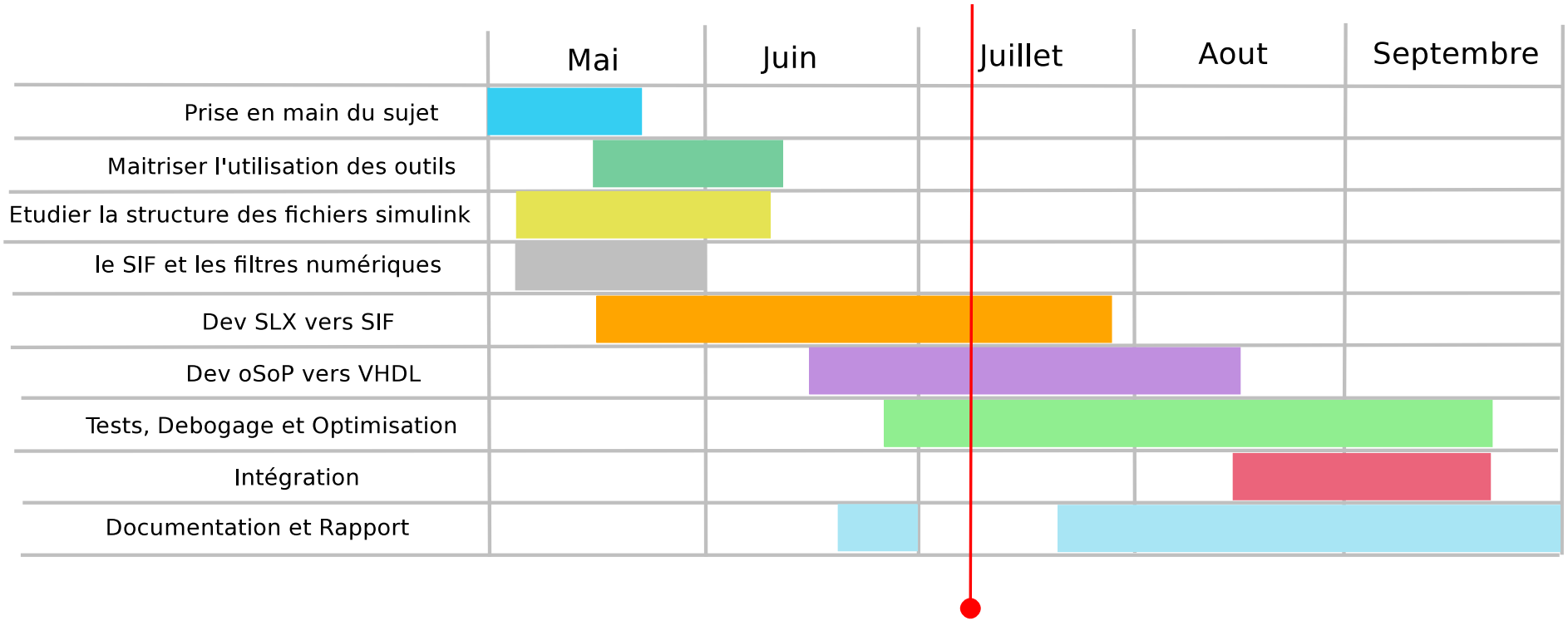
- Essentiellement **par comparaison** au résultat d'interprétation manuelle
- Comparaison au résultat pour le filtre LWDF déjà été fait "manuellement" par un membre de l'équipe PEQUAN
  On doit avoir le **même résultat**
- Petits diagrammes simulink pour tester les fonctionnalités internes

Procédure de recette

Pour la partie utilisant Stratus : **oSoP vers VHDL**

- FiPoGen peut générer un **code C** virgule fixe correspondant à l'oSoP
- Nous, on génère du code **VHDL** à partir de l'oSoP
- On verifiera par **double simulation** :
  On génère aléatoirement les mêmes stimulus pour les 2 codes puis on lance les 2 simulations, ce qui devrait nous donner le même résultat.

Echéancier


Mai
Juin
Juillet
Aout
Septembre
Prise en main du sujet
Maitriser l'utilisation des outils
Etudier la structure des fichiers simulink
le SIF et les filtres numériques
Dev SLX vers SIF
Dev oSoP vers VHDL
Tests, Debogage et Optimisation
Intégration
Documentation et Rapport