Implémentation sur FPGA de la formule de Sherman-Morrison à faible latence pour du matériel dans la boucle

SLIM ASSALI, MICHEL LEMAIRE, JÉRÉMY POUPART, DANIEL MASSICOTTE

Université du Québec à Trois-Rivières, Département de génie électrique et informatique,
3351, Boul. des Forges, Trois-Rivières, Québec, Canada,
Laboratoire des signaux et systèmes intégrés

Résumé – Le matériel dans la boucle (HIL – *Hardware-in-the-Loop*) est de plus en plus présent dans différents domaines de la recherche académique et industrielle, notamment, dans les applications portant sur les réseaux électriques. Dans ce cadre, nous explorons l'utilisation du FPGA (*Field Programmable Gate Arrays*) pour le développement de systèmes HIL à très faible latence. L'objectif est de concevoir et d'implémenter sur FPGA des algorithmes à faible latence adaptés aux contraintes du temps réel. Cette étude intègre l'application de la formule de Sherman-Morrison (FSM) pour la simulation de deux types d'onduleurs triphasés : les onduleurs à deux niveaux (2-L) et la topologie de convertisseur Back-to-Back (B2B). L'objectif est de minimiser la latence des calculs, tout en garantissant une précision conforme aux exigences des applications, en utilisant une représentation en virgule fixe.

Abstract – HIL (Hardware-in-the-Loop) Hardware-in-the-Loop (HIL) is increasingly present in various academic and industrial research fields, particularly in applications related to power networks. In this context, we explore the use of Field Programmable Gate Arrays (FPGAs) for the development of ultra-low-latency HIL systems. The objective is to design and implement low-latency algorithms on FPGAs that meet real-time constraints. This study incorporates the application of the Sherman-Morrison formula (SMF) for simulating two types of three-phase inverters: two-level inverters (2-L) and the Back-to-Back (B2B) converter topology. The goal is to minimize computation latency while ensuring accuracy that meets application requirements, using fixed-point representation.

1 Introduction

La simulation en temps réel des convertisseurs de puissance est cruciale pour valider les lois de commandes dans les microréseaux, les véhicules électriques et dans nombreuses autres applications. En fait, l'approche HIL permet de valider la performance et la fonctionnalité d'un contrôleur sans avoir recourt au système réel. Le risque sur matériel réel est ainsi écarté ce qui permet de réaliser des tests dans des situations normalement destructrices. Dans cet article, la représentation du circuit est basée sur le modèle binaire des interrupteurs de puissance [1]. Dans cette approche, deux valeurs de résistances sont utilisées pour modéliser l'état ouvert ou fermé des interrupteurs. Le circuit est modélisé à l'aide d'une équation linéaire $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$ où $\mathbf{A}$ est la matrice d'admittance $\mathbf{b}$ est constitué des sources de tension et de courant connu dans le circuit et $\mathbf{x}$ des variables de tension et de courant inconnues. $\mathbf{A}$ est une matrice variante dans le temps et le nombre de matrices différentes est fonction du nombre d'états possibles des interrupteurs [1][2]. L'inversion des différentes matrices $\mathbf{A}$ sont traditionnellement précalculées et stockées en mémoire. Or, cette solution devient rapidement impossible pour des circuits complexes en raison du très grand nombre de combinaisons possibles. En effet, il y a 2^N matrices inverses possibles où N correspond au nombre d'interrupteurs dans un onduleur (figure 1). Pour pallier ce problème, nous pouvons tirer profit de la formule de Sherman-Morrison (FSM) afin de réduire la quantité mémoire requise pour les implémentations sur FPGA tout en tentant de conserver une latence faible.

L'article est structuré comme suit : la section 2 présente une reformulation des calculs de la FSM pour le cas des onduleurs, la section 3 décrit les implémentations de la FSM sur FPGA selon une architecture semi-systolique modulaire développée à l'aide des outils Vivado HLS et System Generator, la section 4 présente les résultats d'exécution de la FSM pour des applications HIL à faible latence et finalement une conclusion à la section 5.

2 Reformulation des calculs de la FSM

Une autre écriture de la FSM est proposée dans [3]

$$\left(\mathbf{A}+\mathbf{u}\mathbf{v}^{\mathrm{T}}\right)^{-1}=\mathbf{A}^{-1}-\sigma\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1} \quad (1)$$

où σ est un scalaire exprimé par

$$\sigma=\frac{1}{1+\mathbf{v}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{u}} \quad (2)$$

$\mathbf{A}$ est la matrice de référence sélectionnée, et $\mathbf{u}$ et $\mathbf{v}$ sont deux vecteurs colonnes qui représentent les perturbations appliquées à la matrice $\mathbf{A}$.

Suivant la FSM tel qu'exposé dans [3], la matrice de référence inverse $\mathbf{A}^{-1}$ est stockée afin qu'elle puisse servir de référence pour le calcul des matrices variantes dans le temps. Avec la connaissance de la perturbation $\mathbf{u}\mathbf{v}^{\mathrm{T}}$ à la matrice $\mathbf{A}$, il est possible de calculer rapidement l'inverse de la nouvelle matrice requise pour solutionner l'équation linéaire. Cette méthode est applicable en séparant les modifications en plusieurs perturbations de rang 1. Ceci permet d'itérer la formule jusqu'à ce que toutes les perturbations soient traitées. On peut alors se débarrasser du calcul complet du produit de vecteur $\mathbf{v}$ et ne sélectionner que les colonnes et les lignes ayant des éléments non nuls de $\mathbf{v}$ et $\mathbf{u}$.

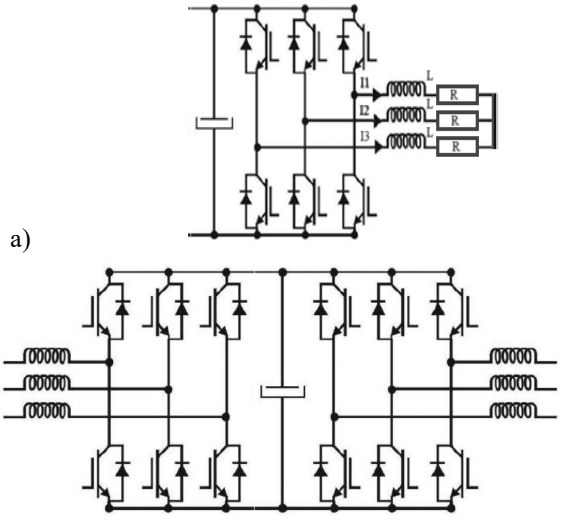

Figure 1 Circuits onduleurs : 2 niveaux (2-L) (a) et Back-to-Back (B2B) (b).

Cela facilite le calcul de σ décrit par l'équation (2) où la quantité $\mathbf{v^T A^{-1} u}$ est un produit scalaire défini par les perturbations sélectionnées par $\mathbf{u}$ et $\mathbf{v}$. On précalcule toutes les valeurs de σ en mémoire afin d'éviter le calcul de la division.

3 FSM sur FPGA – implémentation

Pour l'implémentation, nous comparons deux outils pour évaluer les compromis entre deux approches distinctes: i) Vivado-HLS offre une mise en œuvre rapide, mais avec un contrôle moindre sur l'architecture synthétisée; et ii) System Generator for DSP (SysGen) permet de spécifier une architecture sur mesure, mais nécessite un temps de développement plus long [3][4].

3.1 Implémentation SysGen

L'idée consiste à concevoir un processeur élémentaire capable de satisfaire le calcul de chaque élément conçu de manière à pouvoir manipuler les éléments de la matrice de manière efficace et à effectuer les opérations nécessaires pour mettre à jour les valeurs de $\mathbf{A^{-1}u\ v^T A^{-1}}$ conformément à l'équation FSM. En mettant en œuvre ce processeur élémentaire, il est possible de construire un système qui peut calculer rapidement et avec précision les mises à jour de l'estimation de l'inverse selon FSM de la matrice après une perturbation donnée.

Chaque processeur élémentaire est responsable du calcul d'un élément spécifique de la matrice résultante. En répliquant ce processeur pour chaque élément de la matrice, on crée un système parallèle capable de gérer efficacement des calculs complexes sur des matrices de grande taille. En prenant l'exemple de l'onduleur 2-L, les matrices à inverser sont des matrices de 16×16, tel que présenté par la figure 2 qui décrit notre méthode, où l'on traite chaque colonne itérativement en fonction des perturbations ($\mathbf{uv}^T$). Cette architecture semi-systolique offre une modularité qui permet son adaptation à différentes tailles de matrices, assurant ainsi une flexibilité et une réutilisabilité. L'état des interrupteurs détermine dynamiquement les lignes et colonnes de la matrice de référence à modifier. Le contrôleur, prenant en

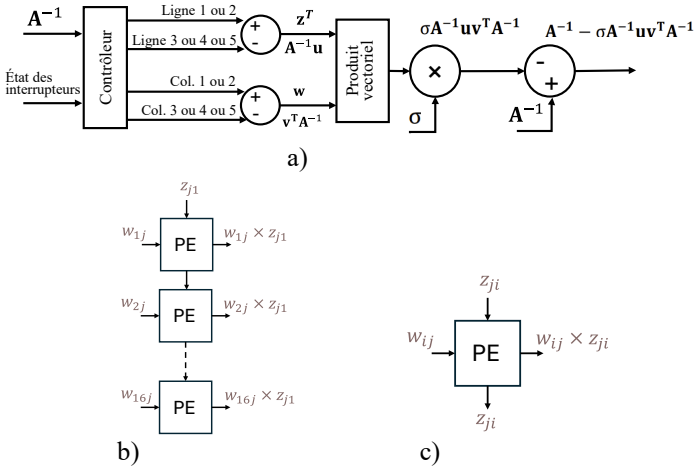

Figure 2 : Les lignes et colonnes provenant de $\mathbf{A}^{-1}$ se propagent selon (a) suivant le produit vectoriel selon (b) utilisant les PE (c).

entrée l'état des interrupteurs et l'inverse de la matrice, identifie les adresses à cibler pour mettre à jour uniquement les éléments nécessaires. Cette architecture, optimisée pour les cas où seules certaines parties de la matrice évoluent fréquemment, permet des mises à jour locales sans recalculer l'inverse complète.

3.2 Ordonnancement de la FSM

L'analyse détaillée des calculs matriciels et vectoriels, effectués itération par itération et interrupteur par interrupteur au sein de la FSM dédiée à notre circuit de puissance, a révélé que le vecteur $\mathbf{v}$ ne contient que des zéros, à l'exception de deux éléments égaux à 1 et -1, positionnés aux indices correspondant aux deux colonnes sélectionnées. Le vecteur $\mathbf{u}$ représente les éléments qui diffèrent entre la matrice de référence et la nouvelle matrice à inverser dans la colonne concernée qui est sélectionnée par le vecteur $\mathbf{v}$ composé de 1 et -1. Le nouvel algorithme est montré à la figure 3. L'idée essentielle est de simplifier l'opération de mise à jour de la matrice en exploitant le vecteur $\mathbf{v}$. En fait, multiplier une matrice par un vecteur rempli de 0 sauf et de deux points de valeur 1 et -1 revient à sélectionner les deux colonnes de la matrice correspondante aux positions des éléments non nuls du vecteur. Il suffit donc de soustraire les deux vecteurs sélectionnés dans la matrice $\mathbf{A^{-1}}$ en parallèle afin de calculer les produits $\mathbf{A^{-1}u}$ et $\mathbf{v^T A^{-1}}$, dont le résultat est un vecteur ligne pour $\mathbf{A^{-1}u}$ et un vecteur colonne pour $\mathbf{v^T A^{-1}}$. Le produit des deux vecteurs nous donne la matrice $\mathbf{A^{-1}u\ v^T A^{-1}}$ qui peut être

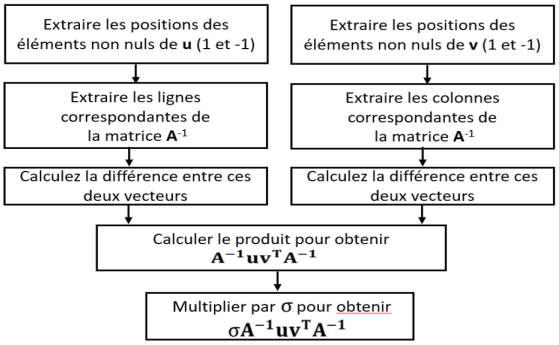

Figure 3 : Organigramme de calcul de FSM.

Tab 1 : Ordonnancement et temps d'exécution de FSM en virgule fixe de 32 bits et virgule flottante simple à 400 MHz.

Étapes	Opérations		Temps d'exécution par itération		Temps d'exécution pour I itérations	
			Virgule fixe	Virgule flottante	Virgule fixe	Virgule flottante
E.1	$\mathbf{A^{-1}u}$	$\sigma\mathbf{v}^{\mathrm{T}}\mathbf{A^{-1}}$	T	4T	T×I	4T×I
			4T	3T	4T×I	3T×I
E.2	$\sigma\mathbf{A^{-1}uv}^{\mathrm{T}}\mathbf{A^{-1}}$		4T	3T	4T×I	3T×I
E.3	$\mathbf{A^{-1}} - \sigma\mathbf{A^{-1}uv}^{\mathrm{T}}\mathbf{A^{-1}}$		T	4T	T×I	4T×I
Délai de synchronisation			2T	2T	2T×I	2T×I
Temps de planification de l'équation de la FSM			**12T**	**16T**	**12T×I**	**16T×I**
Stockage de la matrice de référence			9T	9T	9T	9T
Délai entre les iterations			-	-	5T×(I-1)	5T×(I-1)
Temps total de planification pour la FSM			**21T**	**25T**	**(4 + 17×I)×T**	**(4 + 21×I)×T**
Pour T = 2.5 ns (400 MHz) et I = 6 itérations			**52.5 ns**	**62.5 ns**	**265 ns**	**325 ns**

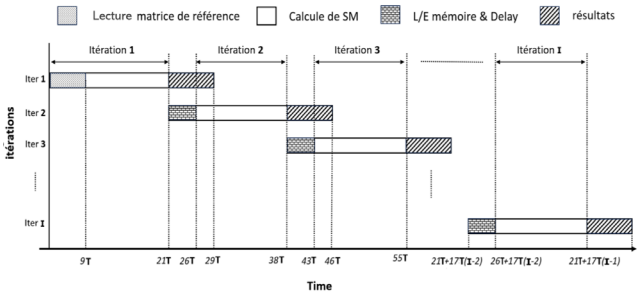

Figure 4 : Séquencements des itérations de calcul de FSM pour l'implémentation en Fixe.

multiplier par sigma pour obtenir la matrice de mise à jour. Cette flexibilité et cette modularité permet d'adapter l'architecture aux matrices de l'onduleur 2-L (figure 1a) aux traitant une matrice carrée 16×16 à partir de quatre blocs 8×8. Ce procédé permet d'obtenir les résultats de l'inverse calculée avec la FSM en un temps équivalent à celui nécessaire pour inverser une matrice 8×8 avec FSM. La figure 4 présente une chronologie détaillée du séquencement de calcul de FSM pour l'implémentation en Fixe. Chaque itération se décompose en plusieurs phases : i) le stockage de la matrice de référence, ii) le calcul de SM, iii) la lecture/écriture mémoire et les délais, ainsi que iv) la production des résultats. Ces phases sont représentées respectivement par des motifs hachurés, des cases vides, des motifs en briques et des hachures diagonales. Le tableau 1 présente le temps les opérations de calcul nécessaire des résultats de FSM à 400 MHz et comparant les temps d'exécution en virgule fixe et en virgule flottante pour les différentes étapes de calcul de l'équation de FSM, ainsi que leur impact sur la planification globale des itérations (I est le nombre d'itérations). La FSM est aussi décomposée en quatre étapes de calculs des sous-équations distinctes, et montre une répartition du temps par opération et par itération.

3.2 Implémentation HLS

Le tableau 2 montre que les différentes configurations binaires entraînent des variations significatives de la latence de calcul en nombre de cycles d'horloge et des ressources nécessaires comparé à l'implémentation en virgule flottante (WL : longueur de mot binaire, Wint : longueur de la partie entière, Wlf : longueur de la partie fractionnaire). Pour FSM Fixe, ces résultats confirment la

Tab 2 : Implémentation HLS de FSM sur le Kintex-7.

Configuration binaire			Latence		Ressources	
Wint	Wlf	Wl	Nbre de Cycles	Horloge (MHz)	DSP48	
					Nbre	%
16	16	32	33	100	656	23
16	32	48	34	100	1556	55
16	48	64	35	100	2112	75
16	64	72	35	100	2516	89
FSM Simple (32 bits)			50	100	410	14

capacité des implémentations à virgule fixe à offrir des performances compétitives 17 cycles en moins par rapport à FSM Simple tout en conservant une précision adéquate en utilisant seulement 9% plus de DSP48 pour une arithmétique fixe à 32 bits et des matrices de 16×16 avec 6 itérations.

4 Résultats et discussions

Les simulations visent à évaluer la performance en termes d'exactitude des calculs numériques, de calcul (latence) et de ressource dans un cas de simulation d'un circuit onduleur en temps réel. Une simulation réalisée avec Simulink est utilisée pour calculer les trois courants triphasés des onduleurs de la figure 1. Cette simulation permet de comparer les erreurs liées au calcul de ces courants à l'aide de trois méthodes différentes pour l'inversion de matrices qui sont :

- Double : l'inverse est calculé en utilisant la décomposition LU dans MATLAB en double précision (virgule flottante 64 bits),
- FSM Simple : FSM en virgule flottante simple 32 bits,
- FSM Fixe : FSM en virgule fixe 32 bits.

4.1 Précision

La figure 5 présente les résultats d'estimations des courants de phase du circuit 2-L pour ces trois méthodes de calculs. Ces résultats de plusieurs cycles présentent une fenêtre de moins de deux cycles. L'analyse des résultats du tableau 3 présentent la comparaison entre les méthodes FSM Simple et FSM Fixe en termes de précision sur *N* (20 000) points de calcul pour les valeurs de courant (I) circulant dans la charge du convertisseur. Les métriques suivantes sont utilisées:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{1}^{N}(i_{Double} - i_{estimé})^2} \qquad (3)$$

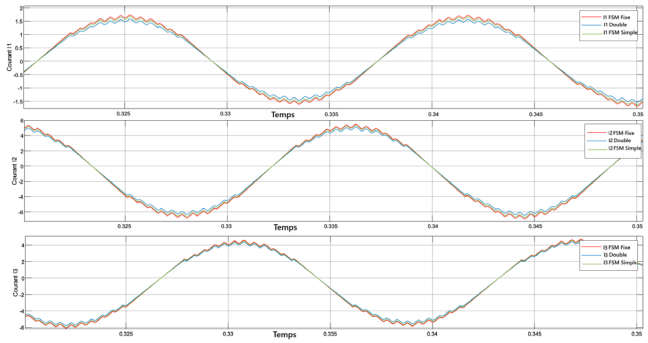

Figure 5 : Estimation des courants I1, I2 et I3 en fonction de la méthode d'inversion pour le 2-L.

Tab 3 : Erreurs calculées des courants du circuit 2-L avec un pas de calcul de 10 µs pour 12 cycles de 60 Hz.

Courant	Méthodes	RMSE	RRMSE (%)
I1	FSM simple	0,0724	6,5
	FSM Fixe	0,11	10,0
I2	FSM simple	0,18	4,0
	FSM Fixe	0,28	7,0
I3	FSM simple	0,11	3,2
	FSM Fixe	0,18	5,0

et

$$\text{RRMSE} = \frac{RMSE}{\sqrt{\frac{1}{N}\sum i_{Double}^2}} \quad (4)$$

Pour I_1, la méthode FSM en virgule flottante simple donne un RMSE de 0,0724 (6,5 % en RRMSE), nettement inférieur au RMSE de 0,11 (10 % en RRMSE) obtenu avec la virgule fixe. Des tendances similaires sont observées pour I_2 et I_3. Ainsi, nous pouvons observer que le calcul utilisant FSM Fixe présente une RRMSE de 10% pour la simulation du convertisseur 2-L, soit 3,5% de plus des courants lorsque comparé au résultat FSM Simple.

4.2 Latence et ressources

Les résultats obtenus dans le tableau 4 montre les performances de l'implémentation de FSM sur FPGA en termes de temps de calcul (nombre de cycle) et d'utilisation des ressources matérielles (DSP48). Si l'on compare HLS (100 MHz) et SysGen (400 MHz) pour l'onduleur 2-L et B2B, on observe des écarts significatifs en termes de temps d'exécution et en ressource. Pour le 2-L, HLS génère une architecture 1,2 (1,5) fois plus lente pour 3,7 (2,3) fois plus de DSP48 que SysGen à virgule fixe (flottante). Pour le B2B, HLS offre une architecture 2,5 (4,7) fois plus lente pour 1,8 (1,5) fois plus de DSP48 que SysGen à virgule fixe (flottante). SysGen permet de mieux exploiter la structure du parallélisme pour minimiser le temps de calcul par un pipeline plus profond utilisant beaucoup plus de cycle d'horloge. HLS génère automatiquement une architecture à partir d'un code C/C++ en ciblant une fréquence d'horloge donnée, avec une optimisation globale du pipeline effectuée par le compilateur. En revanche, SysGen repose sur une construction manuelle à l'aide de blocs Simulink, offrant un contrôle plus précis sur la structure et le pipeline, en revanche plus ardu en temps de développement. Il est nécessaire d'ajuster et de tester l'architecture de manière itérative afin de maximiser sa fréquence (400 MHz).
Cette analyse de performance faite dans le cadre d'un cas pratique de simulation d'un convertisseur 2-L permet d'évaluer l'impact des résultats obtenus pour des simulations temps réel. Cette solution est intéressante pour les applications de convertisseurs, car l'arithmétique à virgule fixe est plus rapide que l'implémentation en virgule flottante sur FPGA. Ceci en fait une approche de choix, notamment dans des contextes où la minimisation des ressources et des pas de calculs sont prioritaire.

5 Conclusion

Ce papier présente une implémentation de FSM sur FPGA selon deux outils, HLS et SysGen, en arithmétique en virgule fixe et flottante. Un résultat quantifié a été présenté selon deux cas d'application : la simulation d'un convertisseur 2-L et B2B. Les résultats montrent que l'implémentation avec SysGen optimise mieux les ressources et atteint une horloge plus rapide pour assurer un temps de calcul plus court par rapport à HLS. Notre implémentation modulaire maximise l'efficacité du processus en favorisant l'évolutivité et la flexibilité, ce qui est bénéfique pour répondre à des cas de matrice de plus grande dimension minimisant le temps de calcul. Rappelons que cette solution FSM demande très peu de mémoire comparativement aux autres méthodes pour la simulation temps réel.

Remerciements
Ce travail a été financé par des subventions du Conseil de recherches en sciences naturelles et en génie du Canada, Prompt, Opal-RT, Hydro-Québec, CMC Microsystèmes et la Chaire de recherche sur les signaux et l'intelligence des systèmes haute performance.

Références

[1] J. Belanger, P. Venne, J.N. Paquin, "What, where and when of real-time simulations," Planet RT, 2010, pp. 37-49.
[2] A. Hadizadeh, M. Hashemi, M. Labbaf, and M. Parniani, "A Matrix-Inversion Technique for FPGA-based Real-Time EMT Simulation of Power Converters," IEEE Trans. on Industrial Electronics, vol. 2, Fév. 2019, pp. 1224-1234.
[3] J. Poupart, M. Lemaire, D. Massicotte, "FPGA Implementation of Sherman-Morrison Formula," Proceeding de la 38e Conference on Design of Circuits and Integrated Systems (DCIS), Malaga, Espagne, Nov. 2023.
[4] M. Lemaire, D. Massicotte, J. Poupart, P. Quinton, S. Rajopadhye, "Polyhedra at Work: Automatic Generation of VHDL Code for the Sherman-Morrison Formula," 14th Int. Workshop on Polyhedral Compilation Techniques (IMPACT), Munich, Allemagne, Jan 2024, pp. 1-10.

Tab 4 : Performance du FSM sur SysGen et HLS en virgule fixe 32 bits et flottante 32 bits sur Kintex-7

	Onduleur 2-L Matrice 16×16, 6 itérations				Onduleur B2B Matrice 26×26, 12 itérations			
Outils	HLS (100 MHz)		SysGen (400 MHz)		HLS (100 MHz)		SysGen (400 MHz)	
Arithmétique	V. fixe	V. flottante	V. fixe	V. flottante	V. fixe	V. flottante	V. fixe	V. flottante
Temps (Nb cycle)	330 ns (33)	500 ns (50)	265 ns (106)	325 ns (130)	1330 ns (133)	3010 ns (301)	520 ns (2080)	640 ns (2560)
DSP48	656 (23%)	410 (14%)	176 (7%)	176 (7%)	1248 (43%)	1064 (38%)	704 (25%)	704 (25%)