Proyecto cálculo de la inversa de una matriz

Curso 2022/23 – Sistemas Empotrados

Universidad de Castilla-La Mancha

Escuela Superior de Informática

Alberto Vázquez Martínez

Alberto.Vazquez1@alu.uclm.es

Explicación de la implementación software

En mi caso, la implementación que he llevado a cabo para el cálculo de la inversa de una matriz cuadrada usa el método de eliminación de *Gauss-Jordan*. Para ello es necesario que dicha matriz sea invertible por lo que se comprueba si elementos en la diagonal no son cero. La matriz de entrada "in" se aumenta con una matriz identidad para formar una matriz mayor "temp". A continuación, se aplica la eliminación de *Gauss-Jordan* a esta matriz "temp", haciendo que el lado izquierdo de la matriz sea una matriz identidad y que el lado derecho sea la inversa de la matriz de entrada "in". Por último, el lado derecho de la matriz "temp" se copia en la matriz de salida "inv".

Para comprobar el funcionamiento utilizo una matriz de 10x10 ya que el cálculo de la inversa requiere muchas operaciones y es muy costoso. Esta es la matriz que utilizaré:

5	3	7	5	1	8	9	4	3	4
6	1	5	4	8	1	1	5	1	5
4	5	3	1	6	8	7	9	8	4
9	6	8	4	7	5	2	6	9	8
7	5	1	5	6	4	9	4	6	4
6	8	4	6	1	8	1	8	4	8
1	2	8	8	7	9	7	9	6	4
9	3	2	6	4	5	9	8	7	5
6	4	8	9	1	3	6	4	9	4
8	4	9	9	3	6	4	8	4	5

La inversa de esta matriz es la siguiente:

1	-0.028897657982453491701	-0.17189028787417122384	-0.09874296413132575374	0.15638110158191761117	0.05284647537111401385 6	-0.09908846417117529637 1	-0.01922596314154774813 4	0.09218309732878606213	-0.18278868213130876026	0.20868584918061490706
2	-0.01644092698439151592	0.10963029147887759381	0.19867083209781385218	-0.17009164313301458771	0.13375306671584880221	0.07555204647513378867 8	-0.14300226322557405984	-0.25729316052453619723	0.13677985145814591767	0.05111778364879348734
3	0.08205344722213106443 34	0.12504224059265908263	0.12502559486554801 22	-0.06418249570935782552 9	-0.07232654130533352832	-0.02827713494140656261 6	-0.08195034259872959895 1	-0.12267726183128840297	0.11903982435824294425	0.00809684424701916228 17
4	-0.08810337794212402797 3	-0.10256935532845876137	-0.16588576231958503947	0.02933049633009511 8656	0.09586266002806502554 8	0.01477430066840406301 7	0.11593252857230818153	0.03502259336077428143 9	-0.04622152542386177257 5	0.06778646043808389420 9
5	-0.05447961073126623648	-0.02377838821632826507 2	-0.04323943304719383637 5	0.07000144636535769063 6	0.12000263134335698369	-0.05342095967920091 05	0.07037596346581574149 3	-0.05341187798213968516 4	-0.09691178582084662087 1	0.05333525819496749759 08
6	-0.01711052195858173913 3	-0.35612976099109129652	-0.23168923871212042301	0.28039288926181177478	0.08463455674043833188 8	-0.07046926777809670185 8	0.18295219782269933949	0.09972771545866592643	-0.32866839852547204984	0.16842927886491039863
7	0.11458719948199815702	0.18968209825554098361	0.13007756480329892603	-0.17825154337204063687	-0.02099521211007381841 4	0.02823933595612858759 5	-0.08994763386098200688 7	-0.04262730485966883921 4	0.16125684930468325725	-0.14369027562555301727
8	-0.03271322121650533603 6	0.18371058374224688159	0.21739271812395997163	-0.20783426367016330202	-0.12959000156766491369	0.04962580735072722776	-0.09670799927062049598 1	0.00112216922105512418 15	0.13635223756148865673	-0.00748620975672978505 85
9	-0.07161539363881395931 8	-0.14415122217872724861	-0.05493731187406688137	0.13796347802216854857	-0.02287774725365279770 5	-0.04578737684304457 14	0.07089442370119489289	0.08670893571637737820 8	-0.00795273510062391476 31	-0.02132845855634783649 95
10	0.14098297171782189767	0.25685014521007171357	-0.03191542118172406059 1	-0.06510999092321342584 1	-0.16511449656267383258	0.17374204284801337049	-0.00779744365141197302 78	0.09095306874032652519 1	0.16370723218277628078	-0.40150490063390884277

Computation time: 0.223 sec.

En mi caso en el código de la implementación hardware especifico la siguiente directiva:

```
#pragma HLS ARRAY_PARTITION dim=1 factor=2 block variable=in
```

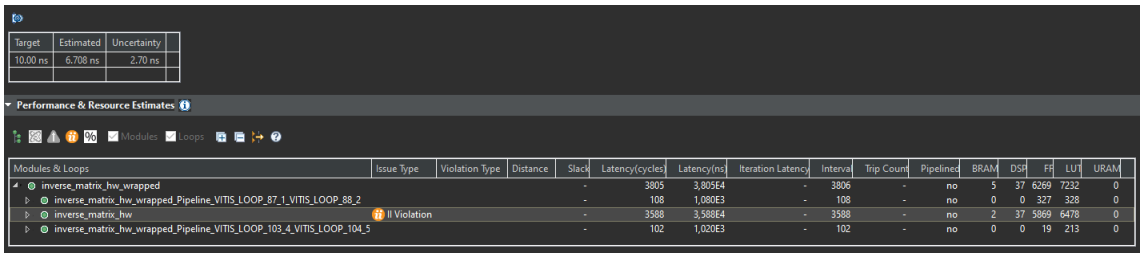
Esto indica como la memoria de la matriz será particionada en la implementación del diseño hardware, en este caso, indico que la primera dimensión del array "in" se particionará en bloques de dos elementos. Este particionado puede ser útil para conseguir un mayor rendimiento o reducir los recursos necesarios para la implementación.

Además, añado la una directiva para realizar un desenrollado del bucle L8:

```
#pragma HLS UNROLL
```

Con esto consigo que el *Interval* (número de ciclos que necesita una iteración) sea de 3806 y el tiempo estimado de ejecución sea de 6.708 nanosegundos.

En el reporte de la cosimulación, vemos como la mayor parte del intervalo se corresponde a los bucles L5, L6 y L7 ya que es donde se realiza la eliminación por *Gauss-Jordan*, ya que las operaciones son muy complejas y además hay dependencias de datos entre iteraciones, por lo que no se puede realizar una desenrollado del bucle para aumentar el rendimiento.



Diseño de bloques en VIVADO

Para el diseño de bloques utilizo 5 componentes:

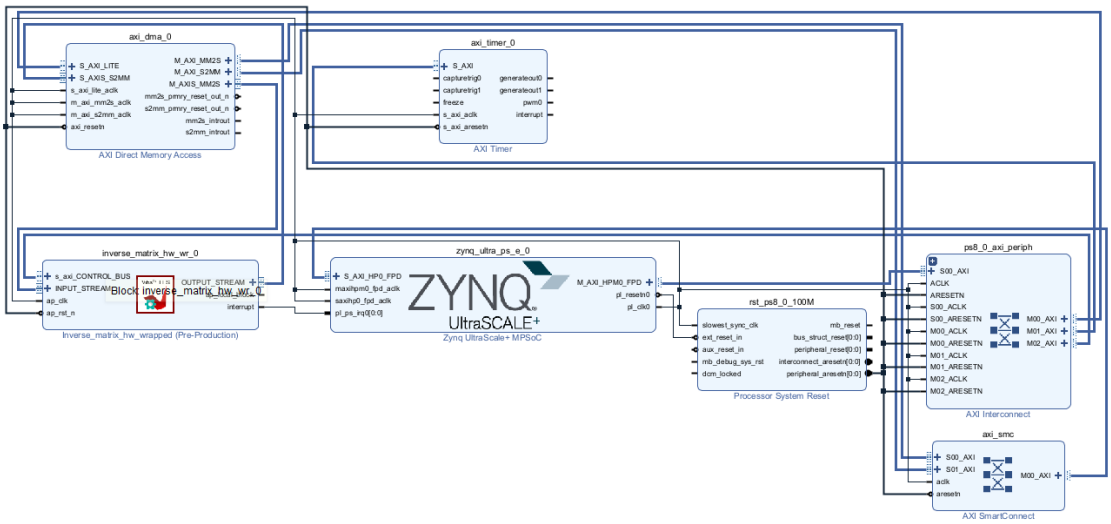
- AXI DMA
- AXI *timer*
- Zynq UltraScale+ PS
- Componente acelerador generado para el cálculo de la matriz inversa

El componente Zynq UltraScale+ PS es el subsistema de procesamiento software. Utilizaré una única interfaz maestra (AXI HPM0 FPD) y una interfaz esclava (AXI HP0 FPD sin coherencia caché). El ancho de palabra lo dejo en 128 bits en ambas interfaces.

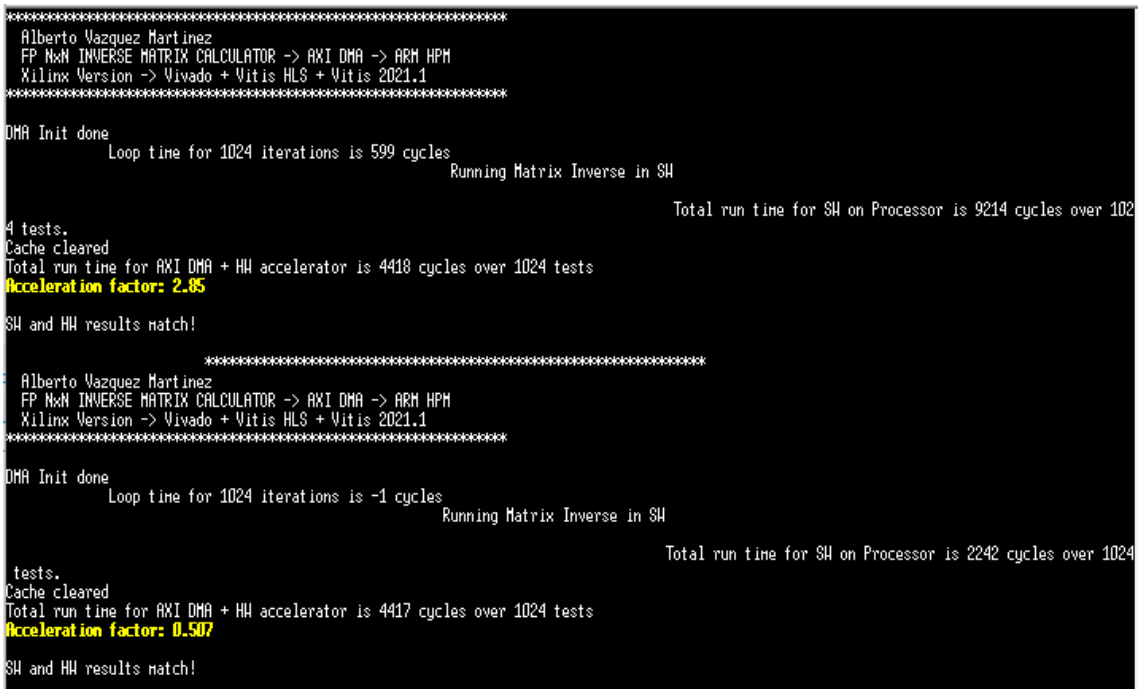
La DMA moverá los datos de la memoria al acelerador. El AXI *timer* se utiliza para genera interrupciones periódicas o medir intervalos de tiempo. Dentro de la DMA, la configuración será la siguiente:

- Deshabilito el *scatter gather engine*.
- *Memory map data width*: 128 bits (al igual que en las interfaces PS-PL)
- *Width of buffer length register* (tamaño del bloque que puede mover la DMA) En este caso utilizo una matriz de 10x10x4 $< 2^9$, por lo que solo necesitaría 9 bits más 1 extra, pero en este caso podré 14 bits.

El diseño de bloques final es el siguiente:



Creando el proyecto de aplicación y utilizando el XSA generado en el programa VIVADO, ejecuto en la placa el proyecto, tanto la versión *Debug* con la *Release*. Como se ve en la imagen, los resultados SW y HW hacen match por lo que la ejecución en placa es correcta.



Para que el match fuese correcto, tuve que cambiar el modo en el que se comprobaban los resultados, ya que, en la versión *Release*, los resultados no hacían match, por lo que la comprobación la realizo de la siguiente manera:

```
/* ************************************************************************** */
//Compare the results from sw and hw
for (i = 0; i < DIM; i++) {
    for (j = 0; j < DIM; j++) {
        if ((inv_sw[i][j] - inv_hw[i][j]) >= 0.00001) {
            err++;
        }
    }
}
```

Código final para el cálculo la inversa de una matriz

```
void inverse_matrix_hw(T in[DIM][DIM], T inv[DIM][DIM]) {
	#pragma HLS ARRAY_PARTITION dim=1 factor=2 block variable=in

	int i,j,k;
	int n = DIM*2;
	float ratio;
	float temp[n][n];

	// Copy in matrix in temp DIM*2xDIM*2 matrix
	L1:for(i=0;i<DIM;i++) {
		L2:for(j=0;j<DIM;j++) {
			temp[i][j] = in[i][j];
		}
	}

	// Augmenting Identity Matrix of Order n
	L3:for(i=0;i<DIM;i++) {
		L4:for(j=0;j<DIM;j++) {
			if(i==j) {
				temp[i][j+DIM] = 1;
			}
			else {
				temp[i][j+DIM] = 0;
			}
		}
	}

	// Applying Gauss Jordan Elimination
	L5:for(i=0;i<DIM;i++) {
		if(temp[i][i] == 0.0) {
			printf("Mathematical Error!");
			exit(0);
		}
		L6:for(j=0;j<DIM;j++) {
			if(i!=j) {
				ratio = temp[j][i]/temp[i][i];
				L7:for(k=0;k<n;k++) {
					temp[j][k] =temp[j][k] - ratio*temp[i][k];
				}
			}
		}
	}

	// Row Operation to Make Principal Diagonal to 1
	k = 0;
	L8:for(i=0;i<DIM;i++) {
	#pragma HLS UNROLL
		L9:for(j=DIM;j<n;j++) {
			inv[i][k++] = temp[i][j]/temp[i][i];
		}
		k = 0;
	}
	return;
}
```