Design-space exploration for high-performance motion control

Pinedo Hernandez, D.S.

Award date:
2013

5T746 Master Thesis ES-E

Design-Space Exploration for High-Performance Motion Control

Dustin S. Pinedo Hernandez
(0759308)
Embedded Systems
d.s.pinedo.hernandez@student.tue.nl

University Supervisors
Kees Goossens
K.G.W.Goossens@tue.nl

Jeroen Voeten
J.P.M.Voeten@tue.nl

ASML Supervisor
Tino Lourens
Tino.Lourens@asml.com

January 26, 2013

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context of the project

ASML is one of the world's leading providers of lithography systems for the semiconductor industry. ASML designs, develops, integrates, markets and services advanced photolithography systems used by customers, the major global semiconductor manufacturers (e.g., Intel), to create chips that power a wide array of electronic, communications and information technology products [1]. With these advanced photolithography systems, ASML helps their costumers to reduce the size and increase the functionality of integrated circuits (ICs). These ICs are etched in silicon wafers which are placed on a motion stage, positioning the wafers under a laser beam.

The motion stage is divided in two mechanical parts, the long stroke (LoS) and the short stroke (SS). The first allows movements up to 2 meters with micrometer accuracy. In contrast, the short stroke permits shorter movements with nanometer accuracy. Therefore, the motion stage can rapidly position a wafer under the laser beam with a nanometer precision. Like other ASML systems, the motion stage has six degrees of freedom (DoFs) permitting three translation movements and three rotational movements. To control this mechanical part a control loop, that is described in Figure 1.1, is implemented.

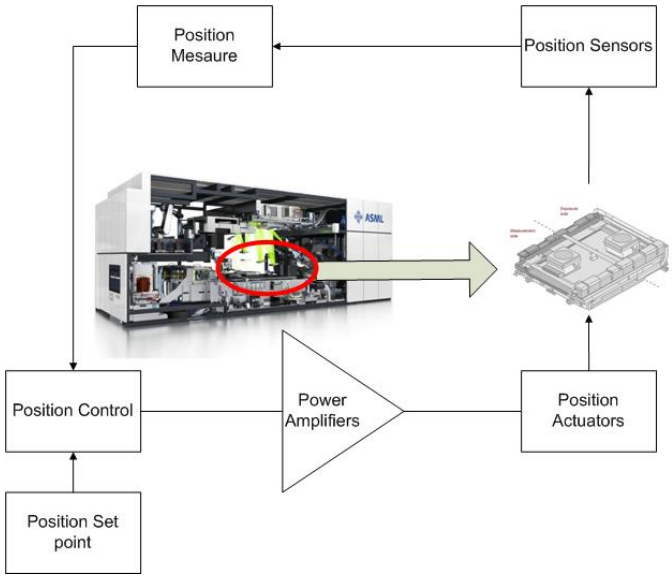


Figure 1.1: Wafer stage motion closed loop

The IC manufacturing industry has to keep targeting Moore's law. This law predicts that an increment in the transistor density will minimize costs. Additionally, the density is doubled every 18 months. To meet these goals and keep a profitable business, IC manufacturing industry must also increment the production rate of chips. Consequently, ASML machines strive for increasingly higher throughput.

To achieve these production goals, the motion stage must move faster and obtain higher accuracy as well. Hence, the number of sensors and actuators must be increased. Consequently, a more complex motion control

with a high sample frequency is being designed targeting future machines. In this change, the current controllers implemented in the short stroke may have to be replaced by a more flexible controllers [3].

The current platform executing the motion control is expected to be insufficient to meet the requirements. To overcome this limitation, it has been proposed to make an step into multicore architectures. This project aims to explore the design-space of multicore where the case of study is a complex motion control with a small input/output (IO) delay and high sample frequency. Additionally, the use of hardware acceleration is proposed in this exploration.

1.2 Problem statement

The main objective of this project is to perform a design-space exploration for high precision motion control. Given a benchmark application of complex motion control developed in a previous project, this project will answer the research question about the IO delay and sampling frequency that can be obtained in a multicore platform.

The main parameters of this design-space exploration are IO delay, speedup and cost. IO delay is defined as the time that takes to give a proper output signal after a new input sample arrives. The speedup is the comparison rate between the design solution sampling frequency and the current motion control sampling frequency which is *10 kHz*, e.g., a 10 times speedup means a design solution sampling frequency of *100 kHz*. The cost is the sum of number of cores and FPGAs. FPGAs are considered in the cost since the exploration considers hardware acceleration.

Two additional parameters are considered to compare the solutions found in the design-space exploration, i.e., applicability and complexity. Applicability refers to the cases where solutions are applicable. Complexity expresses the knowledge that is required to implement the design solutions.

1.3 Approach

To answer the research question, a model-based approach is proposed. To accomplish this goal the CARM2G [2] modeling tools, that have being developed by ASML and the Embedded System institute, are used. This set of tools permits a rapid and easy design-space exploration by modeling solutions and predicting critical issues.

This approach has two main steps, a model calibration and exploration. In the calibration step, CARM2G library is used. CARM2G contains measurement data from most of the application blocks except from the flexible controller which is represented by a state space block. A state space block is a block where the output depends on the input and the current vector state of the block.

To complete the calibration step, the state space block time behavior is measured on a General Purpose Processor (GPP). This timing behavior is stochastic meaning that the best and worst computation times are calculated.

The exploration step focuses on reducing the max execution time among the cores. This reduction should have an impact on the IO delay and the sampling frequency. To achieve this reduction, four strategies are proposed, namely parallelization, decomposition, FPGA acceleration and optimizations. Parallelization aims to execute concurrently heavy computational tasks. Consequently, the load is distributed among more cores. In the decomposition strategy, a heavy computational task is split in smaller tasks that can be executed in parallel. In the FPGA acceleration strategy, a heavy task is deployed on an FPGA reducing its execution time. For this strategy, a recalibration of the model is performed. The optimization technique reduces the execution time of a heavy task by optimizing the task specification.

All these techniques are supported by the use of data flow diagrams which permit to understand the mapping choices. By using these strategies, several design solutions are found in the design-space. These solutions are compared using the five design-space parameters.

1.4 Y-chart paradigm

Y-Chart is a paradigm which separates the concerns of application, platform and mapping to later combine them in a systematic approach, see Figure 1.2. This approach permits a model-based design-space exploration in two steps, "Capture the Past" and "Explore the Future" [5]. In "Capture the past" step, a model close to the system behavior is built. This initial model is the basis of the "Explore the future" step where a performance-driven design-space exploration is conducted. Using multiple metrics of interest and some design parameters, the design-space exploration allows modeling of multiple implementations.

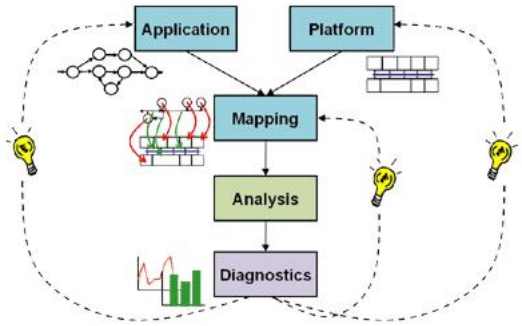


Figure 1.2: Y-chart diagram

In "Capture the Past" step, a performance model of the given benchmark application in a single core is created. This model must accurately represent the existing platform with the application before exploring future solutions. To complete this step, it is essential to first fully model the system.

1.5 Related work

[4] has developed a Vector ASIP processor in an FPGA where the instruction set easily permits to overcome the bottleneck of this complex motion control. This work achieved a $100\ kHz$ sampling frequency with 10μs IO Delay. These performance results will serve as reference for the design space exploration of this project. Moreover, The FPGA measurements of this project are used to recalibrate the model in the FPGA acceleration strategy.

In [6] a separation of concerns where the application and the platform are independently constructed while the mapping is static is presented. This approach has the drawback that new configurations imply new rewiring of the communication channels between platform and application. [7] evaluates the performance of a complex level sensor by modeling. The design-space exploration uses a model-based approach where the concerns are separated with dynamic mapping. To accomplish this the mapping configuration easily switches with minimal effort to different mapping choices. In this work, the application is still aware of which component of the platform requests a service. In [8] a multicore platform study is performed using a similar model approach. The main difference is that the application is unaware of the platform. In this case, the mapping is in charge of the requests.

1.6 Report organization

This report is organized as follows. Chapter 1 gives an introduction to the context and the motivation for this project. The problem statement and the approach are also included in this chapter. Later in Chapter 2, some basic motion control theories are explained. In addition, the characteristics of the given benchmark application are explained in this chapter. In Chapter 3, the modeling tools used and the calibration step which is a preliminary step of the design-space exploration are given. This calibration is focused on one specific block. Chapter 4 contains the design-space exploration steps and a final comparison between solutions.

Chapter 2

Motion Control Theory

Motion control theory is the subfield of control in which the process variable to control is position and/or velocity. Usually, the actuator components are electrical motors. One of the ASML systems is the wafer stage controller which is a motion control system. ASML motion control must have high precision since accuracy is critical and speed determines latency and throughput. The chapter briefly explains the motion control concepts and introduces details about the proposed high performance motion control.

This chapter is organized as follows. First, the basic control concepts are introduced. The mechatronic components of the application are explained in the second part. In the last part the application is detailed, and the performance metrics are presented.

2.1 Digital control

A digital control application is a system which manipulates a process variable. It is called digital because the measurement and actions components communicate with the plant through analog/digital converters and digital/analog converter respectively. In addition, the control action is digital. In Figure 2.1, a general digital control is illustrated. In general, the components which are called blocks are divided in three: measurement, control and action. This set of blocks constitute the control loop.

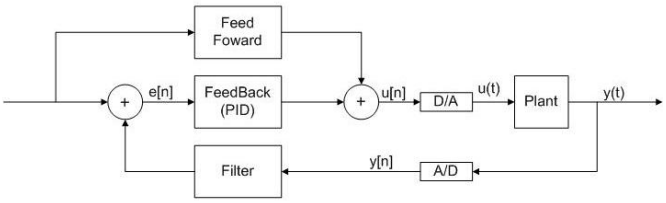


Figure 2.1: Digital control closed loops.

Control loops can be classified as feedback (FB) and feedforward (FF) control loops. The feedforward control loop is considered a manual control because the control block has zero knowledge about how the variable is responding to the control signal. In contrast, the feedback control is considered an automatic control loop since a comparison, between the required value and the current one, is performed. This signal is used as input for the control block. Usually, the FF control loop is added to a system with a FB control loop to improve control performance. This combination increases the control precision over the process variable.

A given high precision motion control design is studied in this project. An abstraction of this high performance motion control can be seen in Figure 2.2. The pre-gain block initially transforms the control signals to the actuators input domain. In the example, the control signal could behave linearly in the interval [0,1] but the actuator input domain could have a polynomial behavior. By this transformation, control blocks which are the most complex in the control loop can be independently designed from the actuators.

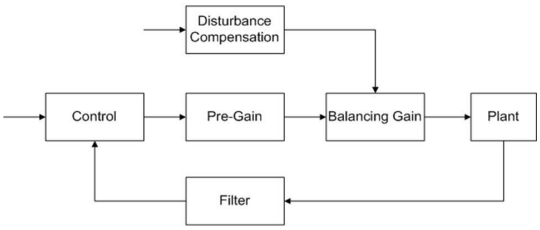


Figure 2.2: High precision motion control.

Complex systems have more than one variable to control. In addition, controlling one variable may affect the behavior of the others. Therefore, a disturbance compensator component which adjusts the control signal has to be implemented. A third component, which is the gain balancing, accordingly modifies the control signals by using this adjustment signal. The final product is a more complex motion controller with higher precision.

2.2 Case of study: benchmark application of the wafer stage motion control

This section presents the case study overview. It focuses in the new block added to the application explaining the complexity of such a block.

2.2.1 Application overview

The case of study of this project is the benchmark application of Wafer Stage Motion Control given in [4]. The wafer stage is the machine responsible for positioning the wafer under the laser beam during the exposure and measurement steps. This machine can move in six degrees of freedom (DoF), three translation movements (X, Y, Z) and three rotational movements (Rx, Ry, Rz) [10]. This module is divided in two subsystems, the Long Stroke (LoS) and the Short Stroke (SS) with non-rigid body (NRB) damping.

The combination of both subsystems permits a rapid positioning of the wafer with a high accuracy. Basically, the procedure is as follows, the LoS provides a coarse positioning in X, Y and the rotation Rz with a micrometer accuracy which is later refined by the SS in six DoF with nanometer accuracy.

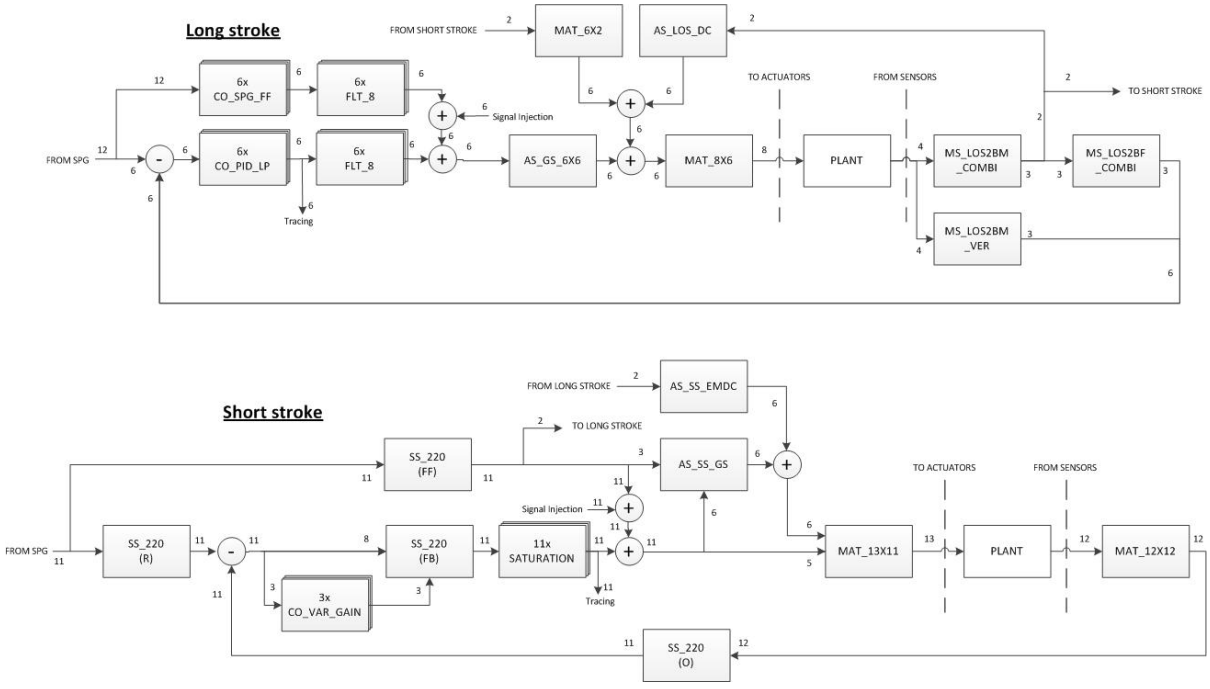


Figure 2.3: Wafer stage motion control.

The block diagrams at the function level of this benchmark application are shown in Figure 2.3 (blocks are described in the Appendix). The numbers in the figure correspond to the number of variables. There are 12 inputs from the set point generator. These inputs correspond to the acceleration and velocity of three position and three rotational variables. The signal injection in the control loop permits to test the motion control to find faults.

There are two important points to denote in this block diagram. First, there is a data dependency between the long stroke and the short stroke which is processed by the disturbance compensator. This data correspond to the current position of the LoS (X, Y and Rz) and the SS control signals for these variables. In the SS, the other variables are analyzed to avoid vibration.

Second, the number of sensors and actuators in the SS stroke roughly duplicate the number of sensor and actuators in LoS. The SS provides a finer positioning than the LoS also movements with six DoF . Therefore, the quantity of information processed by this subsystem is larger. Moreover, a comparison between SS with non-rigid body damping and the current SS have demonstrated that the computational load consumed by the former is 30 times the latter.

This computational load is due to the inclusion of a state space block with a large number of states. This block is further explained next.

2.2.2 SS_220 block specification

The SSB appears in the benchmark application as SS_220 because it has 220 states. A state space block is a system in which the output depends on the inputs and a current state. Usually, the output and the states are related with differential equations. Since the implementation is digital, the differential equations are transformed to difference equations. The difference and the algebraic equations can be written in matrix form. This representation provides a compact methodology to model and to analyze systems with multiple inputs and output. A state space system has the diagram block of Figure 2.4, where A, B, C and D are matrix multiplications.

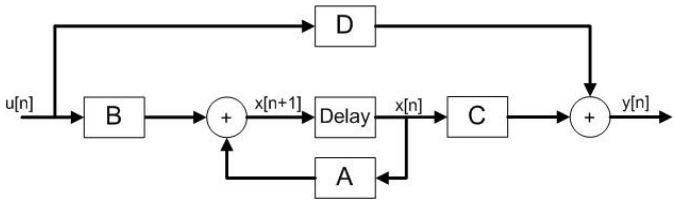


Figure 2.4: State space block diagram

In this figure, *u, x and y* correspond to the input vector, the state vector and output vector, respectively. The state vector is updated in each iteration of the system. This vector has a length of 220. Moreover, the input and output of matrix A are related to this state vector. Therefore, the matrix A is 220x220 matrix. Similarly, matrices B, C and D are 220x12, 12x220 and 12x12 matrices respectively. Hence, the matrix multiplication of A with the states is the largest operation in this block. This operation is the predominant performance bottleneck.

2.2.3 Performance metrics

In the domain of the motion control theory, systems aim to have a rapid reaction with high accuracy. This translates in controller responses characteristics, such as steady state error, damping, settling time and closed loop bandwidth. These performance metrics strongly depend on the chosen controller, sensor and actuator. Since the aim of this project is to evaluate the timing behavior of the motion control, it is unnecessary to evaluate the other metrics.

To analyze the timing behavior of the application two metrics are used: sampling frequency and IO delay. Sampling frequency is the inverse of the sampling period i.e., the number samples taken per second. IO delay, on the other hand, refers to the time from measurement of a variable until a control action is taken. The sampling period and the IO delay have an inverse proportional relation to the performance [11]. Another important remark is to keep sampling period equal to or higher than the IO delay. Otherwise, a new sample will arrive before a control action is taken. This undesirable phenomenon is called overrun.

Chapter 3

CARM2G Modeling Tools

Design solutions can be analyzed using a model-based approach. These models are an abstract but adequate representation of these solutions. CARM2G Modeling Tools are a set of tools which is designed by ASML and the Embedded Systems Institute (ESI). These tools allow to easily model and rapid predict critical issues in the ASML systems.

The purpose of this chapter is to briefly explain the procedure to model the high motion control described in the previous chapter and measure the calibration data for this model. CARM2G Modeling Tools follow the Y-Chart paradigm which is a suitable paradigm for the design-space exploration intended in this project [2].

3.1 Modeling

The system is modeled by separating application, platform and mapping concerns following the Y-Chart paradigm, see section 1.4. The application contains the functionality of the system including the data dependencies between blocks. CARM2G Modeling Tools have three architecture layers resembling the ASML application layers. The bottom layer contains the library of blocks. This library includes the motion control blocks. To support this library, there is a timing behavior profile which includes an interval of measured worst and best computational times of the blocks with an average time.

In addition to blocks, the application layer contains transducers and control loops called servo groups. A transducer corresponds to an abstract representation of an element that connects to either a sensor or an actuators. A servo group refers to a control loop in the application where the blocks and the dependencies, i.e., the connections among them, are described.

Currently, CARM2G Modeling Tools support single-core and multicore platforms. Computation and communication are the two offered resources by the model. The scheduling mechanism is a fixed order scheduler. This schedule is initialized after receiving a synchronization message from an external control unit.

Mapping is responsible for deploying the servo groups according to particular design choices. Each block produces and consumes data. In the case when a consumer block is deployed in a different computational resource than the producer, the mapping would add a communication block on both sides to use the network. Moreover, a block will be executed when all its defined inputs are ready.

The timing behavior of blocks is stochastic. This behavior considers some unpredictable timing responses in processors. Additionally, the model includes a communication penalty. An accurate model of a real time system is obtained by using both computation and communication characteristics.

3.2 Model calibration: state space block

In this section the first step of the approach, the model calibration, is described. The CARM2G library contains measurement data for most of the application block but the state space block. Therefore, the calibration step focuses on this block.

3.2.1 General purpose processor characteristics

To measure the State Space Block (SSB), a General Purpose Processor, with similar characteristics that the single-core processor in the ASML platform, is implemented. Characteristics of this GPP are described in Table 3.1.

Table 3.1: Laptop characteristics

Characteristic	**Detail**
Architecture	Intel Core i5
Number of Cores	2
Hyper-threading	yes
Frequency	2.53GHz
Cache L1	32kB
Cache L2	256kB
Cache L3	3MB
Operating system	Ubuntu 11.10
Compiler	GCC 4.7.0

3.2.2 SSB on benchmark application performance

The SSB is profiled and compared to measurements of the others blocks on the application. For this purpose, the given benchmark application was executed 1000 times, measuring the sampling frequency of the application and the execution time of each block.

These are the results for the sample frequency on the GPP:

- minimum sample period (best case): 1.26 ms (789 Hz)
- average sample period (average): 1.36 ms (731 Hz)
- maximum sample period (worst case): 3.01 ms (332 Hz)

The current motion control executes at 10 *kHz* sampling frequency. In comparison, the motion control of the case of study runs in average 14 times slower and around 30 times slower in the worst case. These results corroborate that this design is more complex than the current motion control. In Table 3.2, blocks with the highest execution times are presented. The SS_220 which is used in the four State Space Blocks (see Chapter 2) takes more than 90% of the application.

Table 3.2: Blocks with high load

Function	**Mean Time (μs)**	**% Load**
SS_220	321,25	91,78
FLT_N	1,875	3,21
AS_GS_6x6	15	1,07
MAT_13X11	15	1,07
Others	40	2.87

3.2.3 SSB optimizations

The execution time of the SSB block is huge than compared with other blocks in the application. To reduce the execution time and improve the application performance, three types of optimizations have been studied for the state space block, which can be discriminated in application, code, and compiler.

Application optimization

There are two application optimizations, a state space transformation, which aims to reduce execution time, and a block split transformation, which reduces the IO delay. A state space representation can be transformed to another state space [14]. Since the matrix A is the heaviest computation in this block, it has been proposed to apply a transformation leading to an upper triangular matrix A. This transformation reduces the multiplications by half.

The second application optimization aims to reduce the IO delay. In Figure 3.1, two executions of the same block for sample n are illustrated. In the top case, the block is executed as one resulting in a sampling period and IO delay of 7 units. In the second case, the block is divided in two sub-blocks, a critical and non-critical block. The sampling period remains the same as 7 units but the IO delay decreases to 2 units.

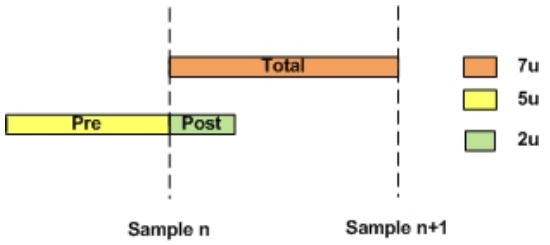


Figure 3.1: Pre-sample and post-sample improvement.

The non-critical block or pre-sample block is a block containing the operations to prepare the original block for the arriving of a sample. These operations are unnecessary to execute when a sample arrives. Instead, they can be executed after output signals of the original block are calculated.

By initially executing the critical block, the IO delay is reduced. To understand how this is applied in the state space block, let us introduce a set of equations which specify the state space block:

$$\mathbf{Y}[n] = \mathbf{Pre}[n] + \mathbf{D} * \mathbf{U}[n] \tag{3.1}$$

$$\mathbf{X}[n+1] = \mathbf{A} * \mathbf{X}[n] + \mathbf{B} * \mathbf{U}[n] \tag{3.2}$$

$$\mathbf{Pre}[n+1] = \mathbf{C} * \mathbf{X}[n+1] \tag{3.3}$$

Where, $\mathbf{Y}$ corresponds to the outputs, $\mathbf{U}$ is the inputs and $\mathbf{X}$ refers to the state vector. The equation 3.1 corresponds to post-sample task which can be executed immediately when an input arrives. Equations 3.2 and 3.3 correspond to a pre-sample task which updates the states for the next input sample. The resulting data dependency is illustrated in Figure 3.2.

This last optimization is already used in other ASML systems and it is supported in CARM2G Modeling Tools.

Code optimization

The benchmark application is implemented in C. In this implementation, the SSB behavior is captured by basic matrix multiplications. Three code optimizations are proposed. These optimizations aim to reduce the overhead of operations and to increase the locality of references. Incrementing the locality increases the speed to access

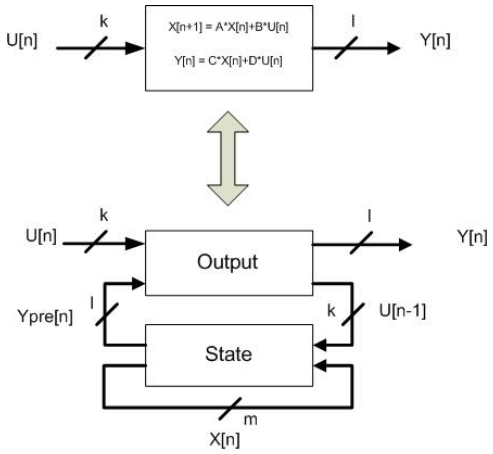


Figure 3.2: Pre- and post-sample task in SSB.

data since data may fit in the cache. Additionally, applications without cache misses have a predictable behavior which is essential in real time systems like the wafer motion control [12].

In Table 3.3, the code optimizations are presented. It can be noticed that the code presented in the table corresponds to a multiplication of the matrix A with the state vector, which is the heaviest matrix multiplication of the block, see Section 2.2.2.

Loop combining refers to merging loops that have the same range. Additionally, it is essential to pay attention to data dependencies between both loops. The data must be consumed after it is produced. The code represents the calculation of a new state vector by multiplying the input with matrix B; multiplying the current vector state with matrix A and adding both results. In the original code, multiplications are calculated separately. However, it is possible to combine the external loop without affecting the data dependencies.

To add results from the matrix multiplication, the corresponding state vector is accumulated. In the original code, this accumulation is stored in its final destination. Therefore, one read and one write to memory are performed for each multiplication result obtained. To reduce this number of memory accesses, a buffer, which is stored in a register, is proposed. This optimization increases the locality of reference and hence the predictability of the application.

The third code optimization is tiling, which splits the operation in smaller independent operations. For a single-core platform the only advantage is that these small operations fit in the cache. Another advantage is an increment on concurrency which could be exploited in a multicore platform. All the mentioned optimizations were applied in all matrix multiplications.

Compiler optimization

The GCC 4.7 Compiler contains optimization flags from O1 to O3. These optimizations aim to reduce code size and execution time. A reduction in execution time has considerable impact in the performance of the application. Thus, O2 to flag will be used to avoid unexpected behavior [13].

One disadvantage of these optimizations flags is the reduction of details in the profiling tool because the application is flattened and blocks are merged. Therefore, the profiler tool is unable to associate a load with a specific block.

3.2.4 SS_220 measurement data

Each optimization described in the previous section has an effect on the performance of the SS_220. The analysis of these performances is presented in this section. However, it is essential to describe the tests first.

Two tests are proposed that aim to evaluate the execution time distribution of the SS_220 in the GPP. The first test measures each iteration execution time creating a spread where the minimum, maximum and mean execution times are calculated. In contrast, the second test measures the total time consumed for the same number of iterations and by dividing both numbers it calculates the average execution time.

Table 3.3: Code optimizations

Optimization	Original code	Optimized code

Loop merging	for (i=0; i < 220; i++) {
	Xd[i] = 0.0f;
	for (k=0; k < 12; k++) {
	Xd[i] += B[i][k] * idt[k];
	}
	}
	for (i=0; i < 220; i++) {
	for (k=0; k < 220; k++) {
	Xd[i] += A[i][k] * X[k];
	}
	}
		for (i=0; i < 220; i++) {
		Xd[i] = 0.0f;
		for (k=0; k < 12; k++) {
		Xd[i] += B[i][k] * idt[k];
		}
		for (k=0; k < 220; k++) {
		Xd[i] += A[i][k] * X[k];
		}
		}

Buffering	for (i=0; i < 220; i++) {
	Xd[i] = 0.0f;
	for (k=0; k < 12; k++) {
	Xd[i] += B[i][k] * idt[k];
	}
	for (k=0; k < 220; k++) {
	Xd[i] += A[i][k] * X[k];
	}
	}
		for (i=0; i < 220; i++) {
		t = 0.0f;
		for (k=0; k < 12; k++) {
		t+= B[i][k] * idt[k];
		}
		for (k=0; k < 220; k++) {
		t+= A[i][k] * X[k];
		}
		Xd[i] = t;
		}

Tiling	for (i=0; i< 220; i++) {
	t = 0.0f;
	for (k=0; k < 12; k++) {
	t+= B[i][k] * idt[k];
	}
	for (k=0; k < 220; k++) {
	t+= A[i][k] * X[k];
	}
	Xd[i] = t;
	}
		for (i=0; i<220; i++) {
		for (x=0; x < min(i+tileRow, n); x++)
		t[x] = 0.0f;
		for (k=0; k < 12; k++) {
		for (x =i; x < min(i+tileRow,n);x++){
		for (y=k; y < min(k+tileCol,n);y++){
		t[x]+= B[x][y] * idt[y];
		}
		}
		}
		.
		.
		.
		for (x=0; x < min(i+tileRow, n); x++)
		Xd[x] = t[x];
		}

The reason to have the second test is to estimate how invasive the first test was and to evaluate how accurate the results are. Let us define the mean execution time as $\bar{x}$, and the average execution time of the second test as $\hat{x}$, then the deviation, σ, is:

$$\sigma = \frac{|\hat{x} - \bar{x}|}{\bar{x}} \tag{3.4}$$

A small deviation corresponds to high precision in measurements.

Original performance and code optimizations

In Figure 3.3, a comparison between the mean execution time after applying the code optimization is illustrated. Note that the execution time increases after applying tiling. It is suspected that the number of memory access has increased. By applying both buffering and combining, a reduction of around 78 μs in the execution time is achieved. This corresponds to more than 25% of improvement, see Table 3.4.

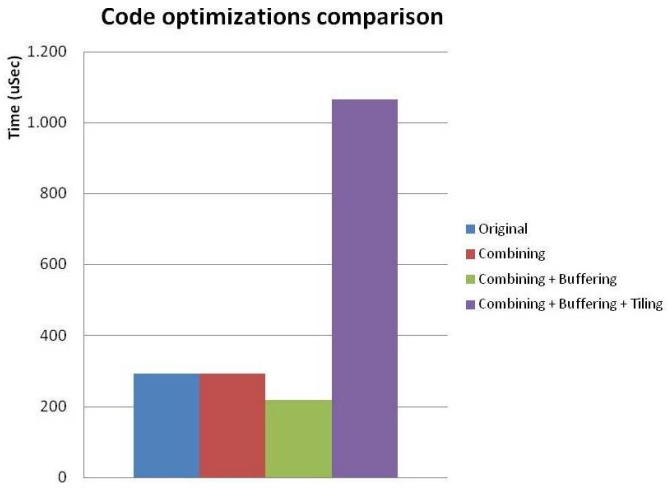


Figure 3.3: Execution time comparison between the three code optimizations.

In Table 3.4, the average is calculated by dividing the total time over the number of iterations which is 1.000.000. Comparing these values with the mean of the spread (frequency distribution), a deviation for this experiment can be calculated. This deviation is small which indicates a high reliability over this experiment.

	Mean. (μs)	**Min. (μs)**	**Max. (μs)**	**Avg (μs)**	**Deviation %**
Original	293.11	289.35	995.62	297	1.31
Comb.	292.96	290.18	902.9	295.98	1.02
Comb. + Buff.	219.24	216.93	867.69	222.82	1.60
Comb. + Buff. + Til.	1065.8	1059.02	3677.4	1062.2	0.34

Table 3.4: Three code optimization results

Compiler optimization and code optimizations

In Figure 3.4, a comparison of the performance of the compiler optimization flag O2 and effects of the code optimization are shown. Using this flag, the execution time is reduced by 80%. The application of tiling increases the performance by almost 8%.

The tiling optimization breaks dependencies in large operations, permitting the compiler to better apply its optimizations. Tiling has additionally increased the predictability of the system, which is demonstrated by the lower deviation obtained in both cases, see Table 3.4 and Table 3.5.

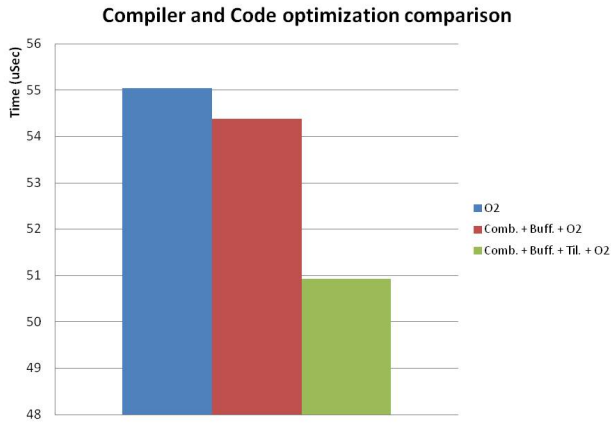


Figure 3.4: Execution time comparison between compiler optimizations.

	Mean. (μs)	**Min. (μs)**	**Max. (μs)**	**Avg (μs)**	**Deviation %**
O2	55.05	53.27	9458.55	54.51	1.00
O2 + Comb.+Buff.	54.38	51.13	309.57	56.98	4.58
O2 + Comb. + Buff. + Til.	51.7	50.44	197.22	50.55	0.73

Table 3.5: Compiler optimizations implementations

Application optimization analysis

Application optimizations were applied to the best result in the previous sections, which is the O2 flag and the three code optimizations. In Figure 3.5 results show that execution time is reduced 40%.

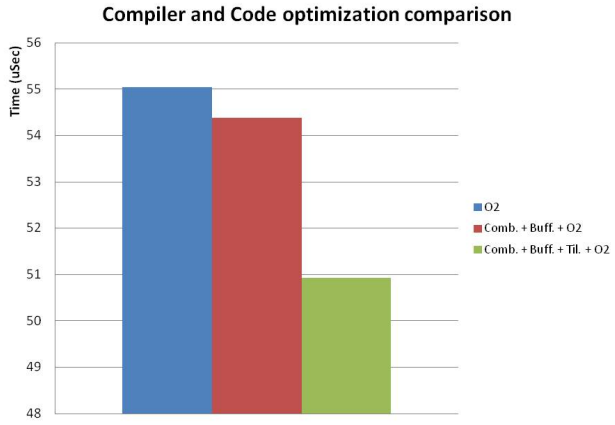


Figure 3.5: Execution time comparison between compiler optimizations.

The execution time of critical blocks or post-sample blocks (see section 3.2.3) is smaller than noncritical blocks. Therefore, a larger reduction in the IO delay is expected. In Table 3.6, these results can be clearly seen. Comparing results from the full and the sum of full pre and full post, it is inferred that an overhead is added by this optimization. However, this is an affordable cost because of the tremendous gain in the IO delay.

	Mean. (μs)	Min. (μs)	Max. (μs)	Avg (μs)	Deviation %
Full	50.92	50.44	197.22	50.55	0.73
Triangular	29.542	29.208	187.772	29.536	0.59
Full pre	51.74	50.545	200.123	55.021	7.05
Full post	0.403	0.389	2.1		
Triangular pre	29.436	29.045	178.333	29.329	1.72
Triangular post	0.397	0.390	2.3		

Table 3.6: Compiler optimizations implementations

Although the triangular transformation permits to greatly reduce the execution time, it could introduce undesirable performance in control metrics such as damping, steady state error or other control metrics which are outside of the scope of this project. Therefore, the full state block will be used.

Calculated times will be fed to the model in the exploration step. These values are:

- maximum pre execution time 200.2 μs
- average pre execution time 51.7 μs
- minimum pre execution time 50.5 μs
- maximum post execution time 2.1 μs
- average post execution time 403 ηs
- minimum post execution time 389 ηs

3.2.5 Model calibration: state space block on an FPGA

Hardware acceleration is one of the possible solutions to explore. Therefore, it is essential to measure the time execution on an FPGA. [4] implemented the State Space block on an FPGA measuring the pre- and post-sample blocks:

- average pre execution time 4 μs
- average post execution time 800 ηs

These measurements will be fed to the model when exploring hardware acceleration in the next chapter.

3.3 Explore the future

The calibrated model serves as a starting point for the design-space exploration. It is unfeasible to explore the whole design-space, consequently logical choices have to be made about the expected region in the design-space where a performance improvement is expected. E.g., adding another computational resource, splitting a task or changing the mapping may improve the application performance.

The independence of application, platform and mapping is the strength of Y-chart paradigm. Each concern can be modified in each exploration step. This independence leads to evaluate the performance of future solutions. This performance can be evaluated using performance graphs and/or Gantt charts.

Chapter 4

Design-Space Exploration of Multicore Platform

In this chapter results of the multicore platform exploration is presented. In the first part, steps to model the current system are depicted. Meanwhile in the second part, the exploration steps and techniques are shown. This exploration results in the comparison graph of Figure 4.1.

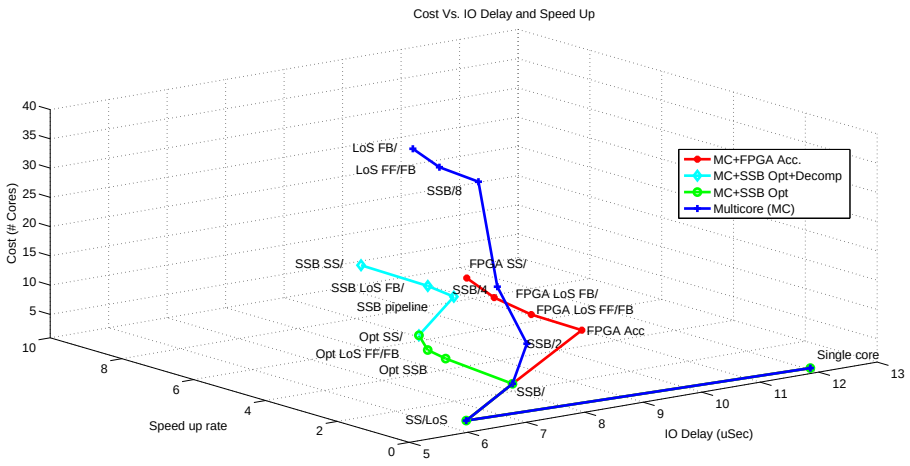


Figure 4.1: Multicore design-space overview

In the graph, the speedup is the rate between the sampling frequency obtained and the sampling frequency for the current motion control which is 10 kHz. The z axis represents the cost which is the number of cores plus the number of processors necessary on an FPGA. The goal is to explore the multicore roadmap and to achieve similar/closest results to the ones achieved on an FPGA which are 100 kHz sampling frequency and 10 μs IO delay [4].

4.1 Modeling

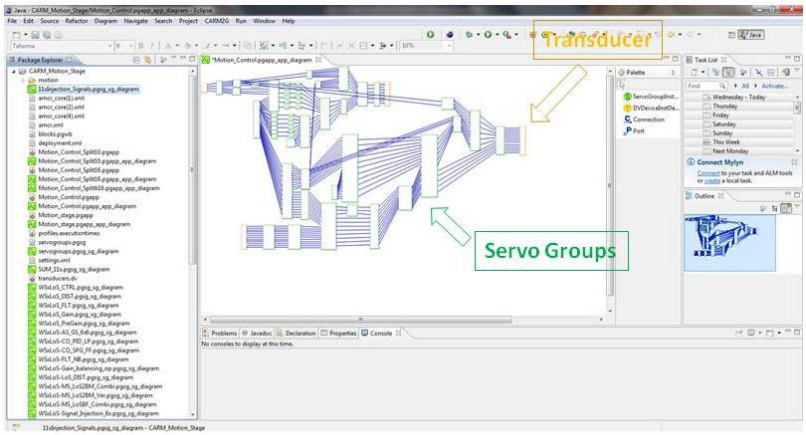
In this section, the steps to create the models are described. Additionally, the model of a single core processor is presented with its performance result.

4.1.1 Model components

The CARM2G Modeling Tools permits to model the given application on different multicore platforms. Calibration values from the standard library and the SSB measurement are fed into the model. These steps result in an accurate model of the given application on the platform.

To built the application, servo groups, transducers and the connections among these elements are described as seen in Figure 4.2. A servo group contains the functional blocks of the system. The calibration data is contained in a profile library of the blocks. a Transducer is an element that communicates with the external world. This element could be a sensor or an actuator.

Sensors produce data to a specific sampling rate. This rate must be smaller than the worst permitted sampling frequency of the system. Otherwise, a second sample will arrive before effectuating the control action. Consequently, this control action may be ineffective.



CARM2G Application Model (PGApp)

Figure 4.2: Application view (PGAPP).

A fixed-order schedule is created with a automatic tool the CARM2G Modeling Tools. These tools currently support deployment of servo groups. Consequently, blocks which belong to the same servo group utilize the same resources. This characteristic limits considerably the number of options in the mapping. To increase mapping possibilities, one single block is described in a servo group. The disadvantage is an expected incremented in overhead.

The data dependency flow is illustrated in Figure 4.3. In this figure, arrows represent dependencies among blocks. The scheduler will try to executed critical blocks first and later noncritical blocks.

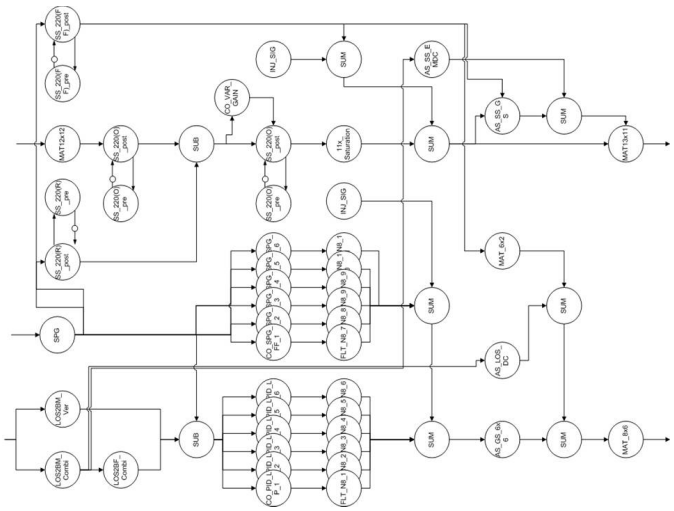


Figure 4.3: Data flow high performance motion control

In the platform which is depicted in Figure 4.4, main elements are depicted. The HPPC is a processor in the architecture which can have from 1 to 32 cores. The control unit is a module that sends synchronization signals other elements, such a clock signal to the External Communication Interface (ECI) unit. This interface unit

communicates between the architecture and transducers. It has high speed links to connect to these external devices. Finally, the SW represents switches controlling the communication traffic in the architecture. Links between elements or boards are SRIO links. Additionally, the model includes a standard communication penalty which is higher between boards than cores. Using these characteristics the first model was built.

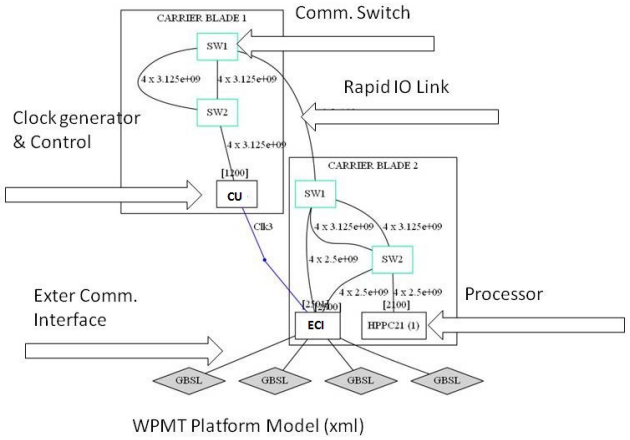


Figure 4.4: Platform view (XML)

4.1.2 Single-core processor performance result

After describing each required component, namely application, mapping and platform; a model of the system is created by the set of tools. This model can be simulated in Rotalumis [16]. Following the simulation in Rotalumis, the performance result can be analyzed by using graphs. The model generates an event file which compiles actions of all boards in the platform (e.g., HPPC, CU and ECI). A Gantt chart can be obtained from this file where the sampling frequency and IO delay can be measured.

TPTview is an official tool of ASML to analyze throughput. It generates Gantt charts to clearly seen the sequence of blocks or task in the system. By using this tool the time behavior, sampling frequency, and IO delay can be comprehended. In Figure 4.5, the resulting Gantt chart for a single-core ATCA rack model is shown.

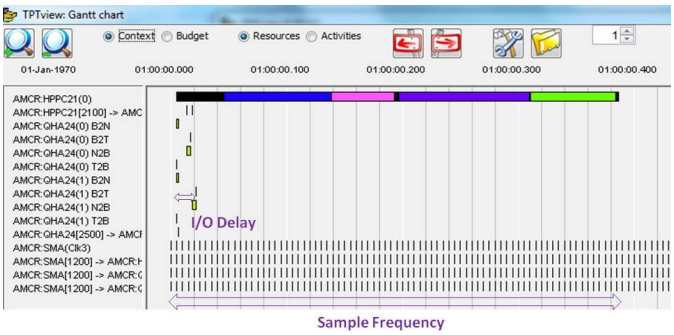


Figure 4.5: Scheduling results in a single core

In this figure the IO delay and the sampling frequency are discriminated. The big difference in these two metrics is that the application only runs post-sample tasks to calculate the output. A post-sample task is a task which is executed after a new sample arrives. However, there are also pre-sample tasks to prepare the system for the next sample which in general have larger execution times, see Section 3.2.3. IO delay depends only in post-sample tasks. In contrast, the sampling frequency depends on both types.

Since the model contains stochastic execution times, the resulting sampling frequency and IO delay will be also stochastic. This behavior was measure using 100 samples as presented in Table 4.1.

Table 4.1: Performance results single core

	Sampling Frequency (kHz)			IO Delay (μs)		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	2.574	3.276	3.768	10.86	12.06	13.57

The obtained sampling frequency is less than the current motion control's sampling frequency of 10 kHz. Remember that the complexity of LoS and SS with integrated non-rigid body damping is 30 times than previous motion control systems. Also, the resulting IO delay have doubled the target value of 6μs. These results put the single core model as the model with lowest cost but also with the lower performance, see Figure 4.6.

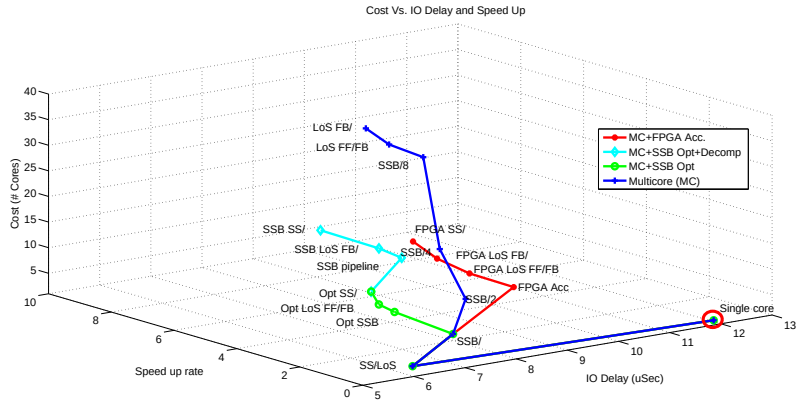


Figure 4.6: Single core solution in the Multicore Design-Space

4.2 Design-space exploration

In this section, the design-space exploration is presented. Strategies during this exploration are fully explained including the consideration take in account.

4.2.1 Strategies overview

To improve the performance of the LoS and SS with NRB damping, multicore platforms have been chosen. During design-space exploration, the aim is to reduce maximum execution of time among all cores which should improve sampling frequency while keeping IO delay as smaller as possible. This reduction is obtained by focusing on the heavy computation load task or bottlenecks. The proposed strategies should be the base for more complex design-space exploration (e.g., minimizing the average utilization).

Data flow diagrams are used to described mapping choices. Therefore, tasks with the same color share resources, e.g., they are executed in the same core. The resulting performance will be compared with the current-old motion control running at 10 kHz and the results on a FPGA of 100 kHz sampling frequency and 10 μs IO delay. As presented in previous section, the single core design executes slower than the current motion control. The question is how the performance can be improved.

4.2.2 Parallelization

The current application contains two control loops in which four blocks consume most of the processor capacity. This design leads to results presented in Table 4.1. In the parallelization technique, blocks or tasks are distributed on different resource to reduce execution time of the application.

In the data dependency of Figure 4.3, several concurrent tasks can be seen. By executing these block in parallel, it can improve the sampling frequency of the system. Usually, an improvement can be predicted if heavy blocks or tasks are executed concurrently.

Parallelization short stroke and long stroke

The first step to improve the performance of this application is to separate the two control loops in the application. Each control loop is mapped to a different core. This can be seen in the data flow of Figure 4.8. The corresponding platform for this model is shown in Figure 4.7. From now on, colors in the Data flow represent different resources (e.g., core). Therefore, tasks with the same color share resources.

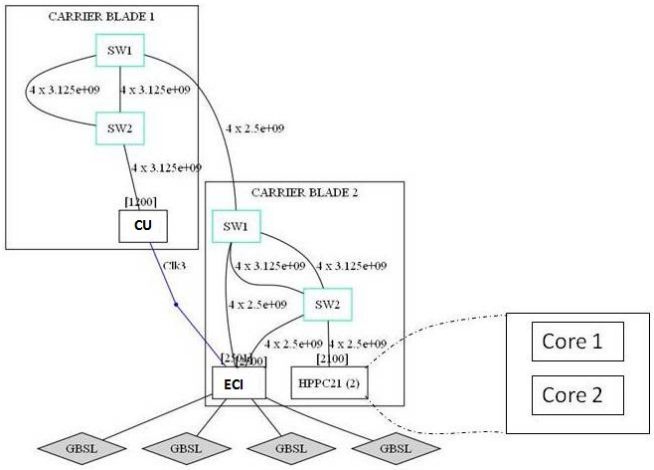


Figure 4.7: Dual core platform

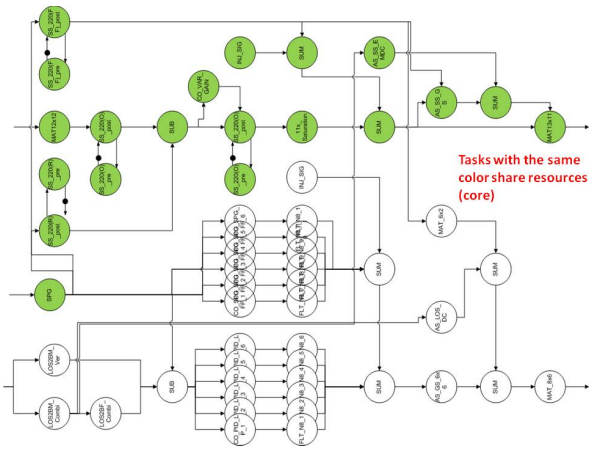


Figure 4.8: Data flow and parallelization of SS and LoS

Performance results are presented in Figure 4.9 and in Table 4.2. There is a high effect in the IO delay when the LoS control loop and SS control loop are separately executed. In contrast there is less improvement in the sampling frequency due the fact that there are four tasks which consume most of the processing time of the Core 1.

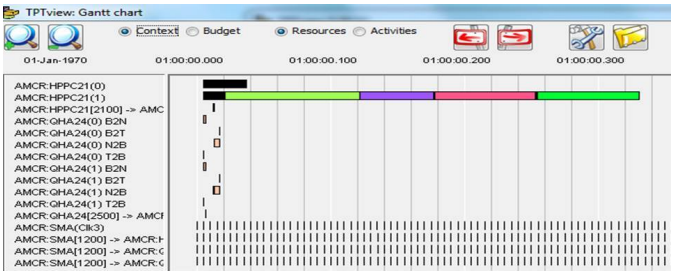


Figure 4.9: Scheduling results parallelization SS and LoS

Table 4.2: Performance results parallelization SS and LoS

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	3.103	3.66	4.44	5.77	6.30	7.26

Parallelization SSB blocks

Most of the execution time of the processors is utilized by pre-sample task of the State Space Block (SSB). By definition, a pre-sample task prepares the block for the next sample, see Section 3.2.3. Consequently, These

tasks only depend on the post-sample task of the same block. Therefore, the pre-sample task of the four SSBs can be concurrently executed. For this model a five cores processor have been described (See Figure 4.10). The corresponding data flow with the mapping is presented in Figure 4.11.

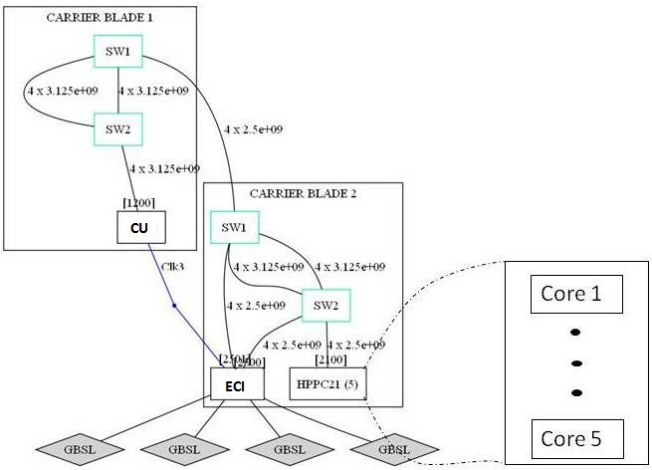


Figure 4.10: Pentacore platform, one core per SSB

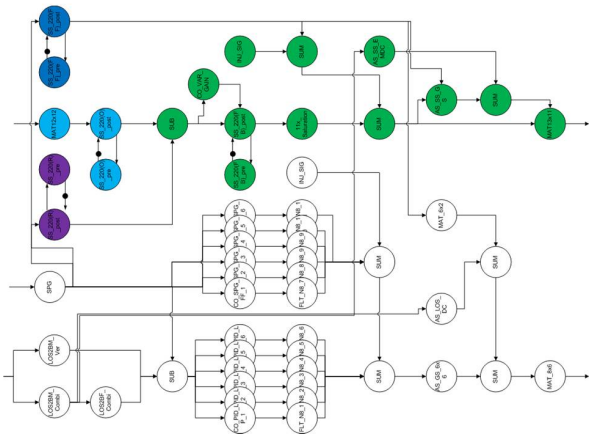


Figure 4.11: One core per SSB

Since each SSB is executed in a different resource, there is a big improvement in the sampling frequency. In contrast, the IO delay suffers for this improvement due the communication between cores. In spite of the separation of these blocks among different cores, SSB blocks are still the main bottleneck of the application (Figure 4.12). Then, a new question arise what it can be done to improve this result.

Figure 4.12: Scheduling results, parallelization of all four SSBs

Table 4.3: Performance results parallelization of SSB

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	5.947	10.419	14.62	6.36	7.41	8.47

Three strategies have been proposed: decomposition, hardware acceleration and SSB optimization as it is illustrated in Figure 4.13. The first strategy splits a heavy computational block (e.g., SSB) in smaller blocks which are easy to parallelize. In the second strategy, a heavy block is deployed in an FPGA to accelerate its execution. Meanwhile, the third strategy proposes to reduce the execution time by optimizing the application itself.

4.2.3 Strategy 1: multicore - decomposition of state space Block

4.2.4 strategy overview

Decomposition splits a heavy computational load task in smaller tasks. This splitting of a tasks that can be concurrently executed. Currently, the heavy computational block is the SSB which has two tasks, the pre- and

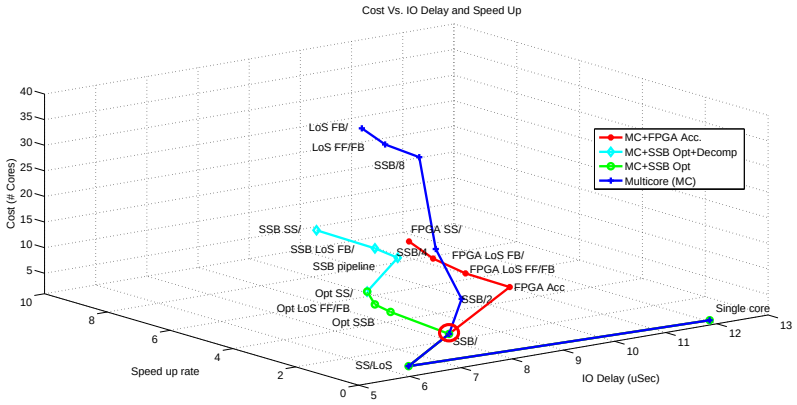


Figure 4.13: Multicore design-space, SSB bottleneck

the post-sample tasks. Between these two, the pre-sample has the biggest computation load.

In the previous chapter, the specification of the SSB was presented. Additionally, an optimization technique called tiling was introduced. By implementing tiling, it is possible to decompose this block permitting a more fine grain deployment as mentioned. Applying tiling permits to transform the Equation 4.1 into Equation 4.2.

$$\mathbf{X}_{n+1} = \mathbf{A} * \mathbf{X}_n + \mathbf{B} * \mathbf{U}_n \tag{4.1}$$

$$\begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \mathbf{X}_3 \\ \mathbf{X}_4 \end{bmatrix}_{n+1} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \mathbf{A}_{1,3} & \mathbf{A}_{1,4} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & \mathbf{A}_{2,3} & \mathbf{A}_{2,4} \\ \mathbf{A}_{3,1} & \mathbf{A}_{3,2} & \mathbf{A}_{3,3} & \mathbf{A}_{3,4} \\ \mathbf{A}_{4,1} & \mathbf{A}_{4,2} & \mathbf{A}_{4,3} & \mathbf{A}_{4,4} \end{bmatrix} * \begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \mathbf{X}_3 \\ \mathbf{X}_4 \end{bmatrix}_n + \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \mathbf{B}_3 \\ \mathbf{B}_4 \end{bmatrix} * \mathbf{U}_n \tag{4.2}$$

Using this last equation, the single pre and post data dependency graph can be transformed into a new data dependency graph as illustrated in Figure 4.14. This data flow mostly corresponds to the dependency described in Equation 4.2. Remember that this equation corresponds only to heavy operations of the pre-sample task. While, the data dependency includes both pre- and post- sample tasks.

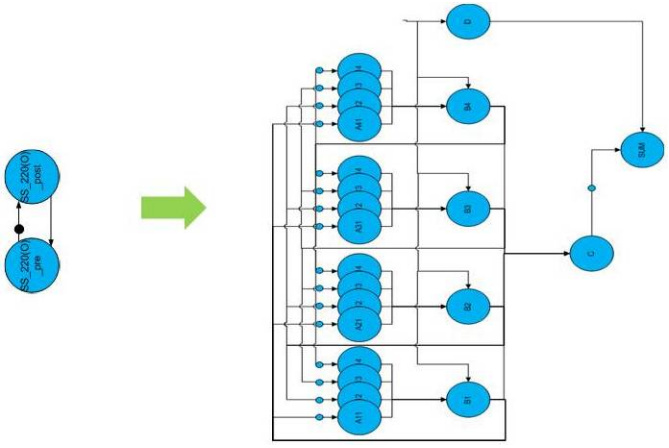


Figure 4.14: New data flow SSB

Parallelization of SSB in 2 set of tiles

Parallelization is possible to apply again with the splitting of SSB. Then, half of SSB tasks are mapped in a different resource. Since there are four SSBs, the number of cores rises from 5 to 9. The modified data dependency is illustrated in Figure 4.16. Additionally, the corresponding platform is given in Figure 4.15.

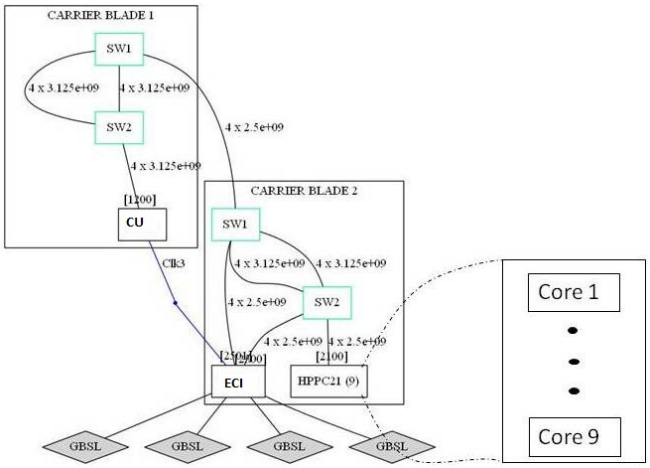


Figure 4.15: Nine cores platform

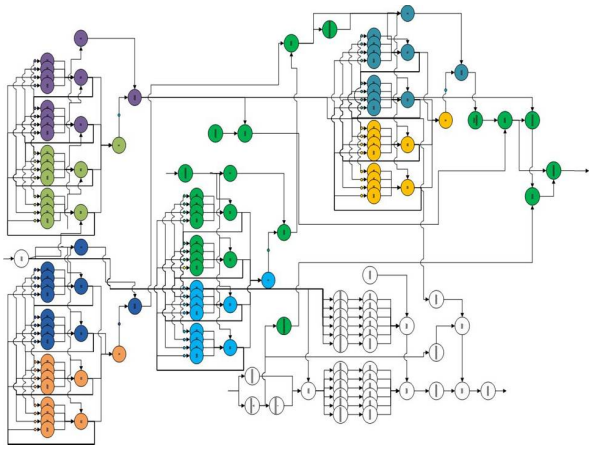


Figure 4.16: Data flow after decomposed each SSB by two

The parallelization in half almost duplicates the previous sampling frequency, see Table 4.4. In contrast, the IO delay increases since more core to core communication is required. In the Gantt chart, see Figure 4.17, it is visible that Core 1 where there is a SSB is still the bottleneck of the application. Therefore, it is necessary to increase the number of resources (cores) to run more tiles of the SSB concurrently.

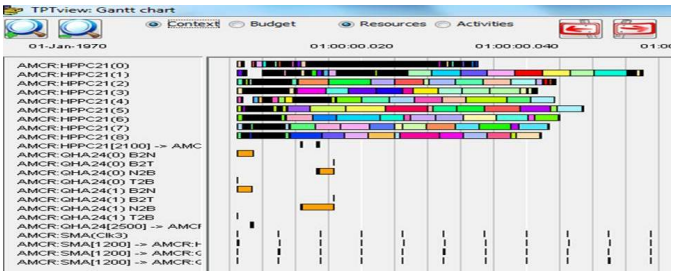


Figure 4.17: Scheduling results, SSB split in 2 set of tiles

Table 4.4: Performance results SSB split in 2 set of tiles

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	15.007	19.026	20.834	7.21	8.19	9.8

Parallelization of SSB in 4 set of tiles

In this step, each SSB is executed in four cores. Therefore, the number of cores needed rise to 17. To clarify, additional Short Stroke blocks are executed in the same core of one set of tiles of the SSB(FB). The LoS control loop is executed separately in an extra core. Four SSB in four cores plus one core for the LoS gives in total 17 cores, see 4.18.

Results are presented in Figure 4.20 and Table 4.5. Similar to previous approaches, decomposing the SSB increases the sampling frequency at the cost of increasing the IO delay. Since the SSB in Core 1 is still the bottleneck, SSBs are even more decomposed. Due to Core 0 and Core 1 execution times are quite close, Core 0 will be the next bottleneck.

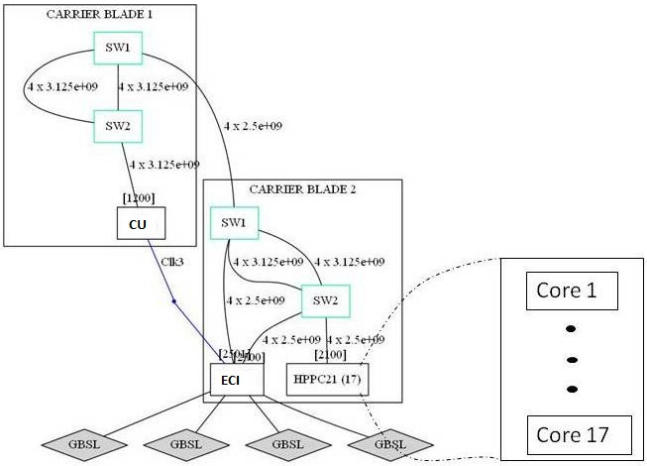


Figure 4.18: Seventeen cores platform

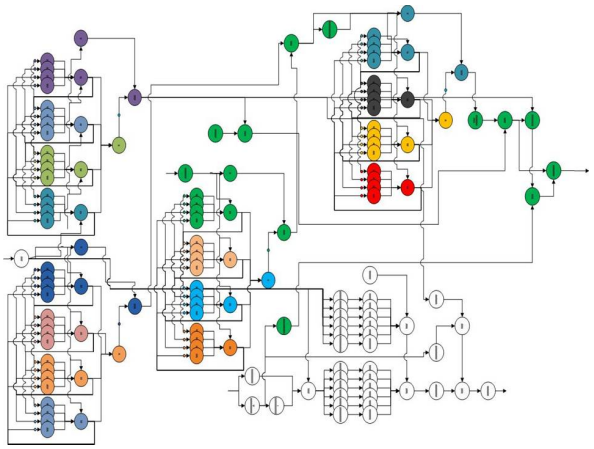


Figure 4.19: Data flow after decomposed each SSB by 4

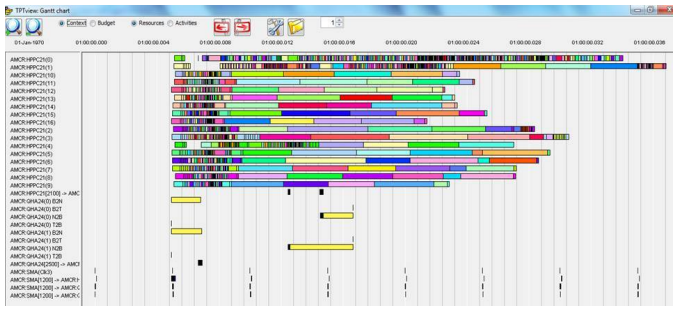


Figure 4.20: Scheduling results, SSB split in 4 set of tiles

Table 4.5: Performance results SSB split in 4 set of tiles

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	24.561	28.233	31.364	7.42	8.24	9.5

Parallelization of SSB in 8 set of tiles

In this model, each SSB is deployed in 8 cores. Four SSB per 8 cores plus one core to executed the LoS control loop and another to executed the SS control loop give a total number of cores of 34. Assuming a limitation of 32 cores per processor, it becomes necessary to add a second HPPC board to the platform, see Figure 4.21. The question is how many cores are needed.

To avoid huge communication overhead, one SSB is migrated to this second processor. Remember that each SSB has 11 inputs and 11 outputs; In contrast it has 220 states. If only pieces of the SSB are mapped in this new processor, then mayor communication traffic could be expected.

The SSB(FF) is migrated to the HPPC22 since it is a non-critical (See section 3.2.3) block and it runs in parallel to the SSB(FB). The mapping for this model is represented in the data flow graph of Figure 4.22.

Results of this model are presented in Figure 4.23 and Table 4.6. After applying this design, a new bottleneck arise. As it can be noticed in the Gantt chart, the bottleneck is Core 0, as predicted in the previous section, which runs the LoS control loop. Therefore, the concurrency of this control loop has to be exploited to achieve a higher frequency.

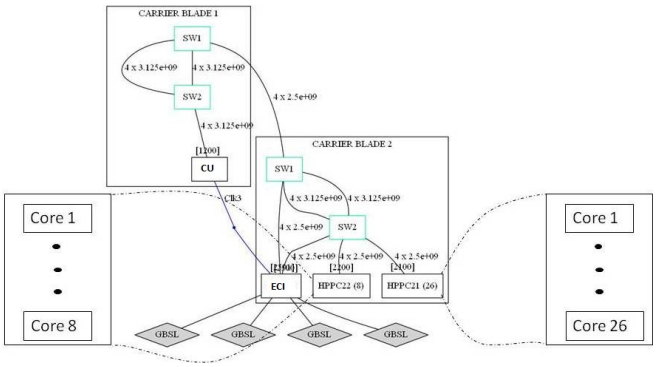


Figure 4.21: Two multicore processors platform, 26 and 8 cores

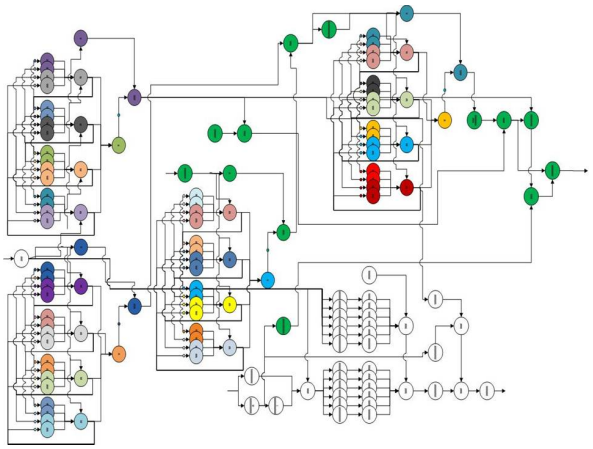


Figure 4.22: Data flow after decomposed each SSB by 8

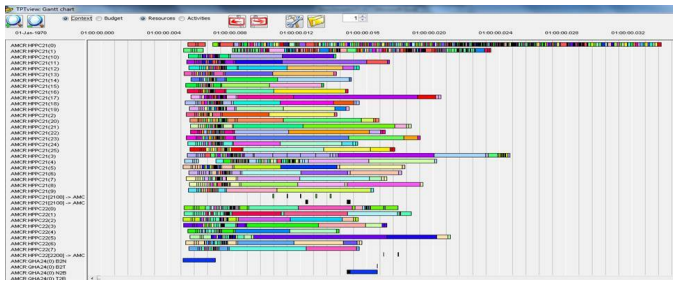


Figure 4.23: Scheduling results, SSB split in 8 set of tiles

Table 4.6: Performance results SSB split in 8 set of tiles

	Sampling Frequency (kHz)			IO Delay (μs)		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	31.643	33.489	35.033	7.15	8.24	9.88

Parallelization LoS FF and FB control

The previous model showed that the LoS is the current bottleneck in the system. To exploit the concurrency of this control loop a second resource is assigned. The FF controller of the LoS is mapped in this new core. The platform for this model and the mapping are presented in Figure 4.24 and Figure 4.25 respectively.

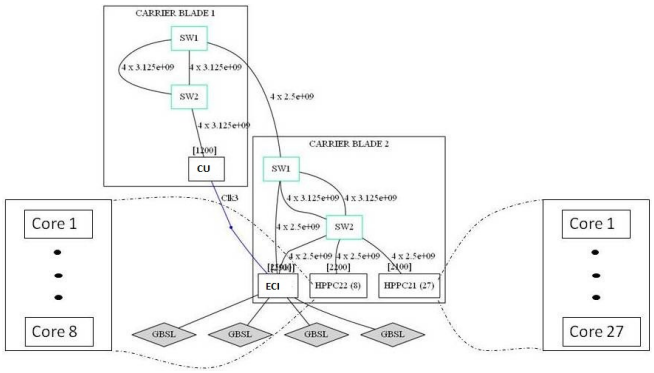


Figure 4.24: Two multicore processors platform, 27 and 8

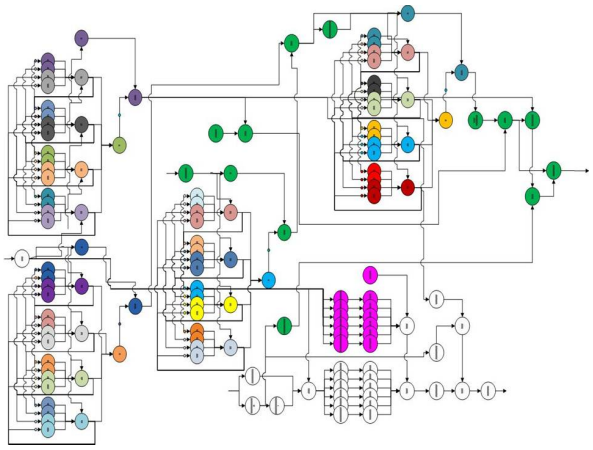


Figure 4.25: Data flow, parallelization of LoS FF and FB

This design increases considerable the sampling frequency without affecting much the IO delay. These results can be seen in Figure 4.26 and Table 4.7. Nevertheless, the LoS control loop is still the bottleneck. Therefore the concurrency of this control loop has to be exploited even more.

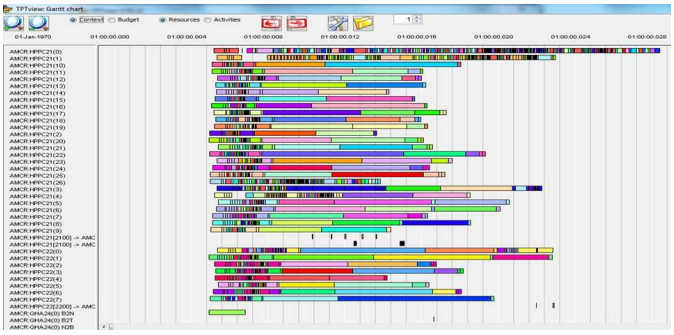


Figure 4.26: Scheduling results, two multicore processors LoS FF and FB

Table 4.7: Performance results LoS control loop parallelization of FF and FB

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	40.206	42.742	44.725	7.35	8.14	9.96

Parallelization LoS FB

Core 0 in the HPPC21 is where the most of the blocks of the LoS control loop are executed. Among these computational blocks, there is the FB controller which has six degrees of freedom. Splitting this controller in two may reduce the load of this core. Therefore, an increment in the sampling frequency is expected. The platform and mapping of this model are presented in Figure 4.27 and Figure 4.28.

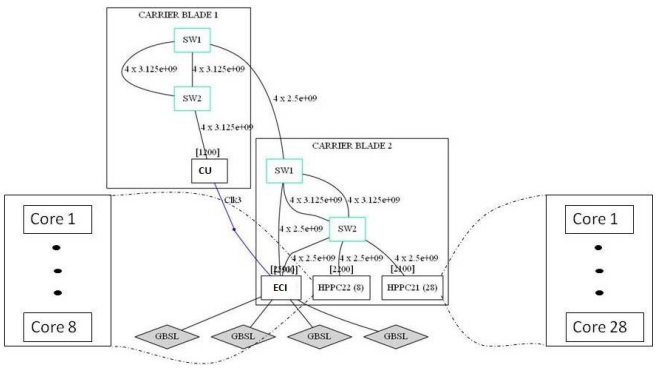


Figure 4.27: Two multicore processors, 28 and 8 cores

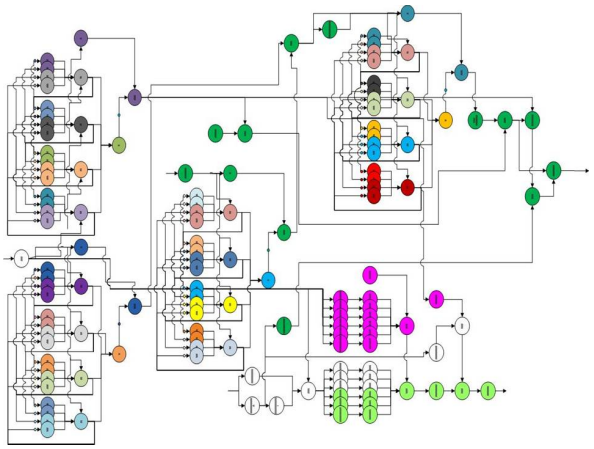


Figure 4.28: Data flow two multicore processors LoS FB parallelization

By applying this configuration, results shown in Figure 4.29 and Table 4.8 are obtained. This design permits to achieve a speedup of five times at the current implementation which is an important result.

In Figure 4.30, this final result is compared with the other solutions in the design-space. The multicore strategy permits to increment sampling frequency without degrading to critical values the IO delay. However, the cost to do so is the higher among three strategies.

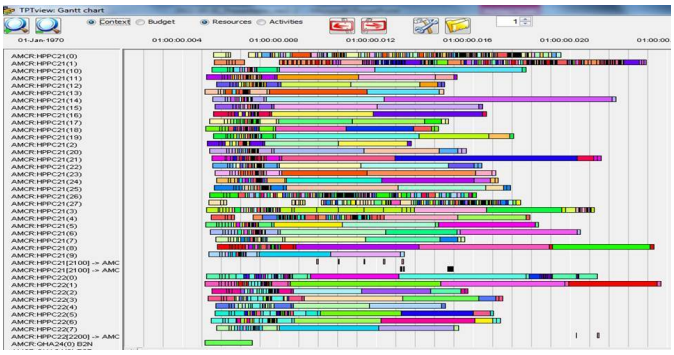


Figure 4.29: Scheduling results two multicore processors LoS FB parallelization

Several bottlenecks appear after this design. To deal with them a decomposition of 16 over SSB is required but also the concurrency of the Short Stroke has to be exploited. Both techniques have been introduced in previous models. In next sections, two different strategies from decomposition are presented to deal with SSB bottleneck.

Table 4.8: Performance results LoS control loop FB controller parallelization

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	45.059	53.074	57.441	7.49	8.32	9.3

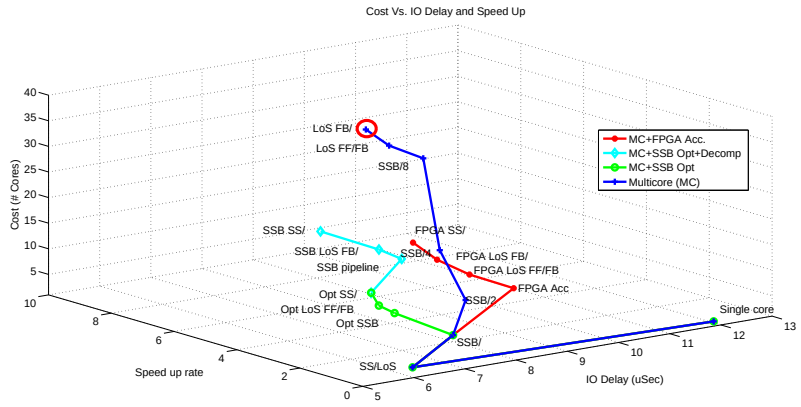


Figure 4.30: Model 8 cost Vs. IO delay and sampling frequency

4.2.5 Strategy 2: FPGA acceleration

Strategy overview

In the previous models, the bottleneck of the SSB is solved by decomposing it and deploy it in more cores. Another solution for this bottleneck, see Section 4.2.2, is to use hardware acceleration. To do so each a SSB is executed in a different resource (e.g., an FPGA) from the LoS and SS control loops. Explicitly modeling of FPGA in the tools is currently not supported. Thus, it is proposed to assume that one core represents one processor designed in [4] which is a vector ASIP built on an FPGA.

The calibration data for this model is given in the previous chapter, see Section 3.2.5. In Figure 4.31 the platform for this model is illustrated. Additionally, the corresponding data flow is shown in Figure 4.32,

By adding hardware acceleration, a speedup of 3 can be easily achieved. In addition, the current bottleneck is the LoS core which is deployed in Core 0. In contrast, the IO delay is higher than in the first strategy. This huge

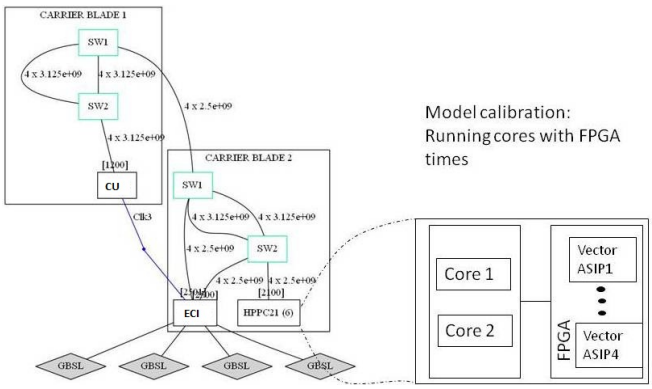


Figure 4.31: Dual core and FPGA acceleration platform

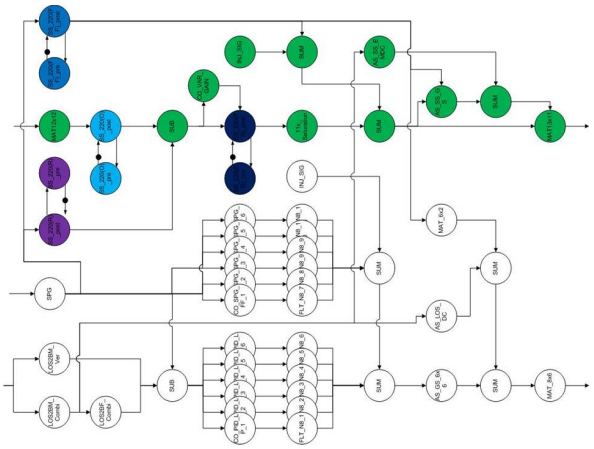


Figure 4.32: Data flow dual Core and FPGA acceleration

difference is due the fact that the post-sample task on an FPGA is twice the execution time the measurement obtained in the GPP, see Section 3.2.5.

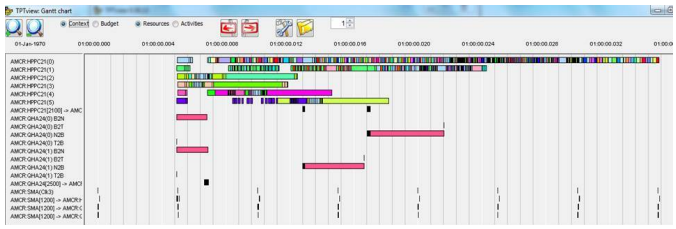


Figure 4.33: Scheduling results, dual core and FPGA acc.

Table 4.9: Performance results, dual core + FPGA acc.

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	30.613	32.855	34.713	9.2	9.92	11.65

Parallelization LoS FF and FB control

Similar to previous steps, the LoS FF and FB are separated in different resources to increase the performance of the design. For this purpose a new platform and mapping are introduced in Figure 4.34 and Figure 4.35 respectively.

Performance results of this model are shown in Figure 4.36 and Table 4.10. As previous optimizations, while the maximum computational load is decreased. The sampling frequency increases. Since the LoS control loop is still the bottleneck of the application, in the next step the concurrency of this control loop is exploited even more.

Table 4.10: Performance results FPGA acc. LoS splitting of the FF controller FB controller

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	43.206	45.354	47.785	8.88	9.74	11.06

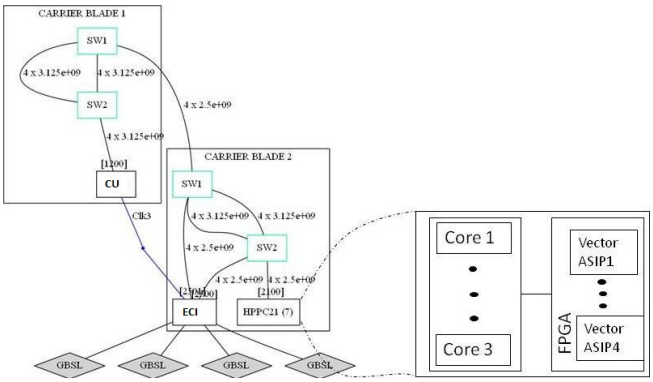


Figure 4.34: Tri-core + FPGA acc. platform

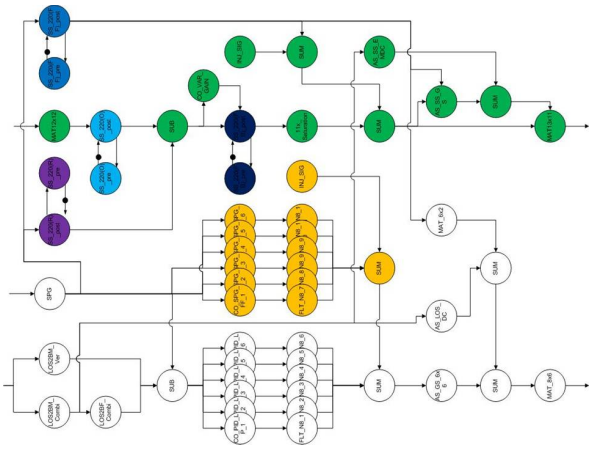


Figure 4.35: Data flow tri-core + FPGA acceleration

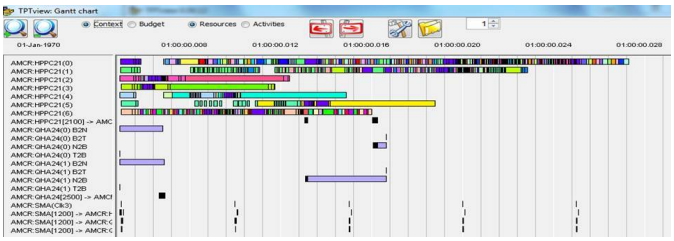


Figure 4.36: Scheduling results, FPGA acc. Parallelization LoS FF and FB

Parallelization LoS FB

Following a previous step, the concurrency in the LoS FB controller is exploited. Since this controller has six DoFs, half of the elements are deployed in a different resource (i.e., core). The platform and the mapping are illustrated in Figure 4.37 and Figure 4.38.

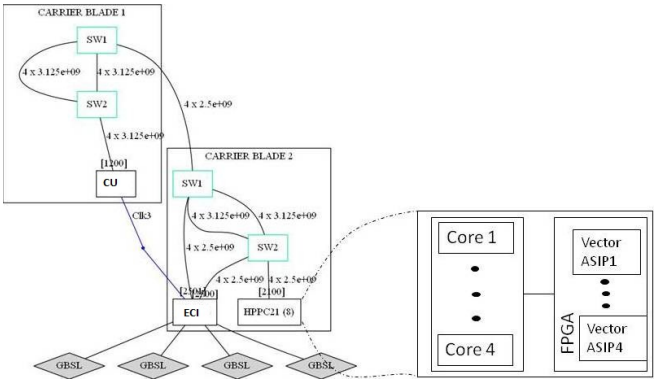


Figure 4.37: Quadcore and FPGA acceleration platform

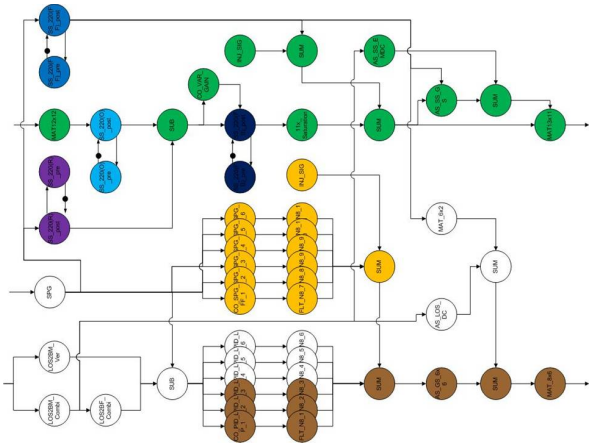


Figure 4.38: Data flow Quadcore and FPGA acceleration

Results of this model are presented in Figure 4.39 and Table 4.11. Currently, the model permits to speedup five times the motion control application without incrementing much the IO delay. In the Gantt chart, it can be observed that the current bottleneck is the SS which is deployed in the core 1. The last step is to decreased this bottleneck.

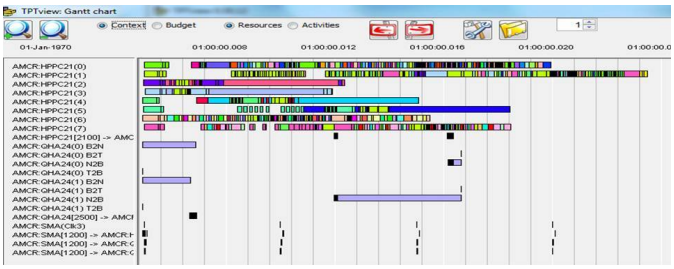


Figure 4.39: Scheduling results, FPGA acc. Parallelization LoS FB

Table 4.11: Performance results FPGA acc. splitting of LoS FB controller

	Sampling Frequency (kHz)			IO Delay (μs)		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	50.734	53.942	56.370	9.07	9.73	10.85

Parallelization Short Stroke

There are other blocks besides the SSB that have pre-sample tasks. These blocks in the SS control loop are concurrently executed to reduce the bottleneck in core 1. The platform and the mapping are shown in Figure 4.40 and Figure 4.41.

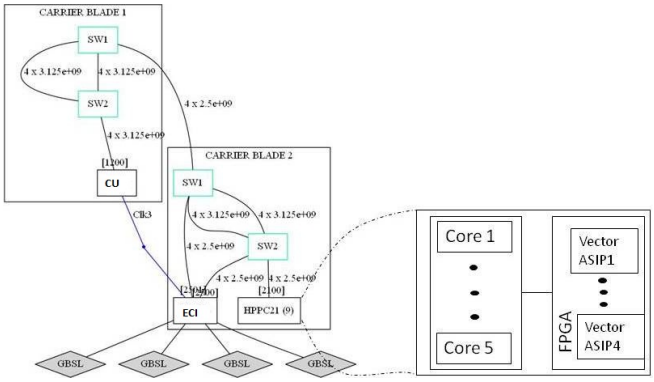


Figure 4.40: Pentacore + FPGA acceleration platform

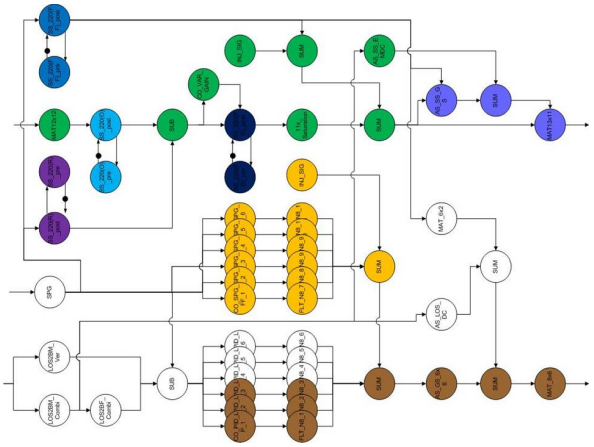


Figure 4.41: Data flow Pentacore + FPGA acceleration

With this mapping a speedup of more than 6 is achieved. Performance results are illustrated in Figure 4.42 and Table 4.12.

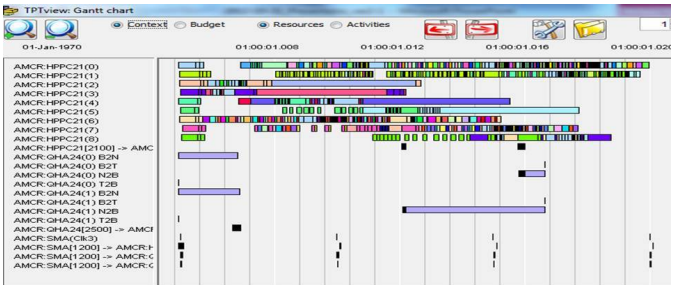


Figure 4.42: Scheduling results, FPGA acc. SS parallelization

Table 4.12: Performance results FPGA acc. splitting of the SS control loop

	Sampling Frequency (kHz)			IO Delay (μs)		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	62.219	64.779	67.089	9.37	9.96	11.24

In Figure 4.43 the comparison between this final model and the whole space is illustrated. The red circle indicates the current model place in the design-space. In the next section, a new approaches is introduced.

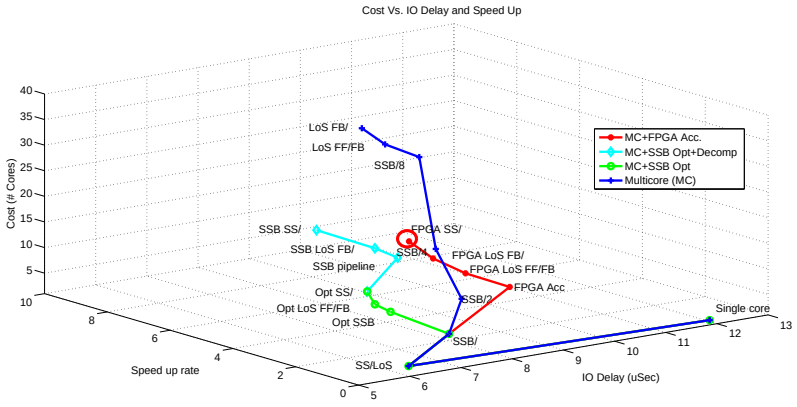


Figure 4.43: FPGA acceleration: cost vs IO delay and sampling frequency

4.2.6 Strategy 3: State Space Optimization

Strategy overview

So far, two strategies over the general cases of the state space block bottleneck were applied. The third strategy uses the look ahead transformation technique to reduce the computational complexity of the SSB. Applying look ahead transformation over equations 3.1 and 3.2 produces equations 4.3 and 4.4

$$\mathbf{X}_{n+1} = \mathbf{A} * \mathbf{C}^{+} * \mathbf{Pre}_n + \mathbf{B} * \mathbf{U}_n \tag{4.3}$$

$$\mathbf{Pre}_{n+1} = \mathbf{C} * \mathbf{A} * \mathbf{X}_n + \mathbf{C} * \mathbf{B} * \mathbf{U}_n \tag{4.4}$$

Note that the number of operations has increased. Though, matrices A, B, and C are constant therefore operations between these matrices can be calculated beforehand. This transformation permits to simplify the matrix 220x220 in the system to a matrix of 220x12, $\mathbf{A} * \mathbf{C}^{+}$. Consequently, measurement values for the SSB on the GPP described in Chapter 3 changes to:

- average pre execution time 9.76 μs
- average post execution time 403 ηs

The execution time of the pre-sample task is reduced by more than 80%. In contrast, the post-sample task remains unchanged. The reason is because the transformation was applied to the specification of the pre-sample task. This optimization is only valid if C has a pseudo-inverse.

SSB Optimization performance

Using recalibration values for the SSB and the same platform and mapping of Section 4.2.2, results shown in Figure 4.44 and Table 4.13 are obtained.

Figure 4.44: Scheduling results, SSB optimization

By applying this optimization, the SSB is still the heavier computational task. Though, the bottleneck of the systems is the LoS. In previous sections, it has been demonstrated how to deal with this bottleneck. The next step follows the previous models to deal with LoS bottleneck.

Table 4.13: Performance results SSB optimization.

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	27.398	32.526	34.574	6.77	7.62	8.93

Parallelization LoS FF and FB Control

To deal with the LoS bottleneck, a separation of the FF and the FB controller is proposed. The mapping and platform are shown in Figures 4.45 and 4.46.

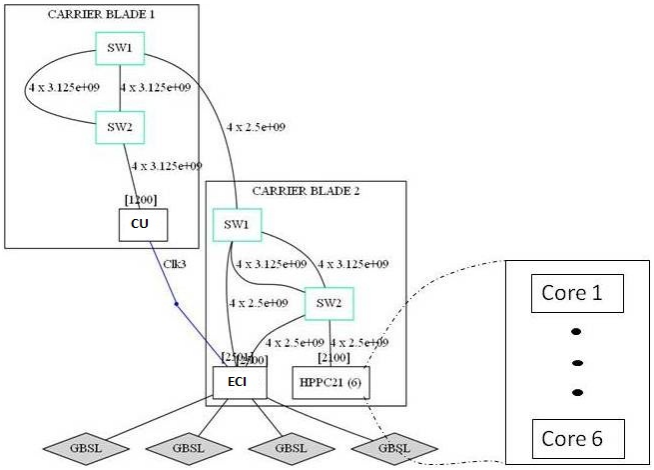


Figure 4.45: Hexacore platform

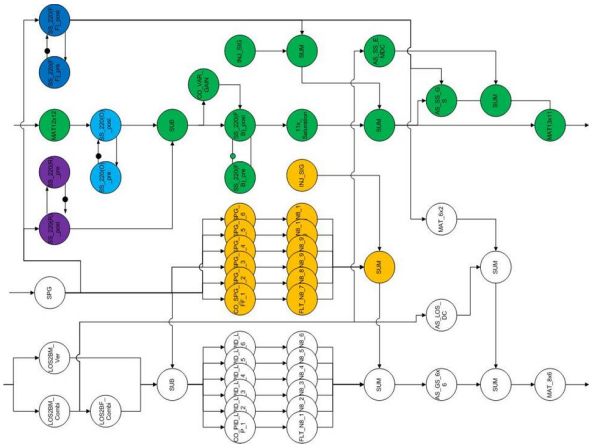


Figure 4.46: Dataflow and mapping SSB opt. LoS FF/FB

Results of this model are presented in Figure 4.47 and Table 4.14. This design solution achieves between 3-4 times the speedup of the current motion controller without affecting negatively the IO delay.

The current bottleneck in design is the core 1 which the SS and on SSB is executed. In the next model, the SS is separated in concurrent tasks.

Parallelization SS control loop

In this model, a new resource is added to the design to decrease the bottleneck of the Short Stroke. Additionally, blocks related in the same sequential path of a correspondent SSB share resources with this SSB. This is clearly seen in Figure 4.49. The platform of this design is illustrated in Figure 4.49.

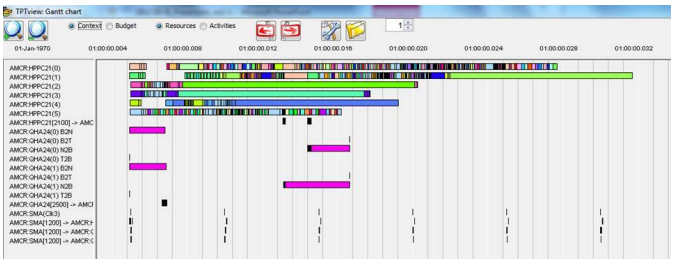


Figure 4.47: Scheduling results, SSB opt. LoS FF/FB

Table 4.14: Performance results SSB opt. splitting of LoS FF and FB controllers.

	Sampling Frequency (kHz)			IO Delay (μs)		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	29.131	36.191	40.303	6.79	7.52	8.49

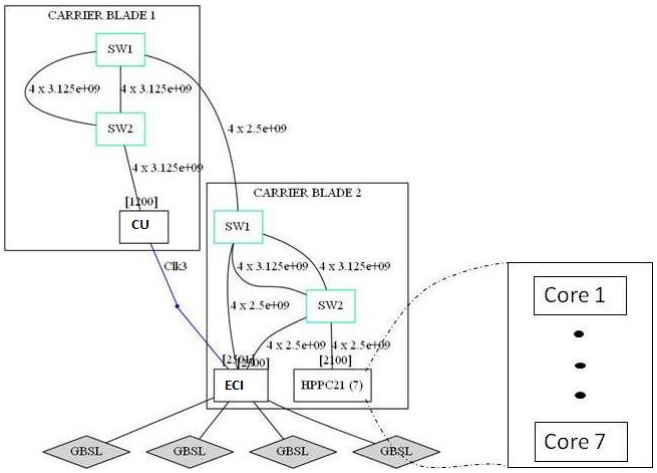


Figure 4.48: Heptacore platform

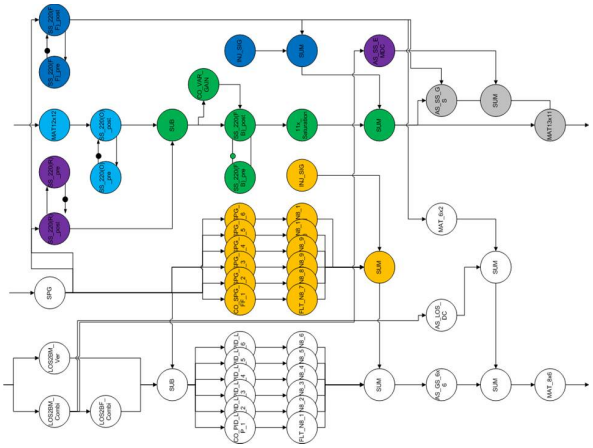


Figure 4.49: Data flow and mapping SSB opt. SS parallelization

Results for this model are shown in Figure 4.50 and Table 4.15. This solution permits to speedup the current motion control 4 times.

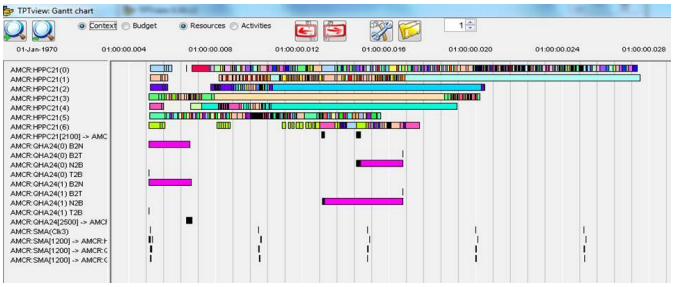


Figure 4.50: Scheduling results, model 15

Table 4.15: Performance results SSB opt. splitting of SS control loop

	Sampling Frequency (kHz)			IO Delay (μs)		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	33.489	42.254	46.969	6.81	7.76	9.65

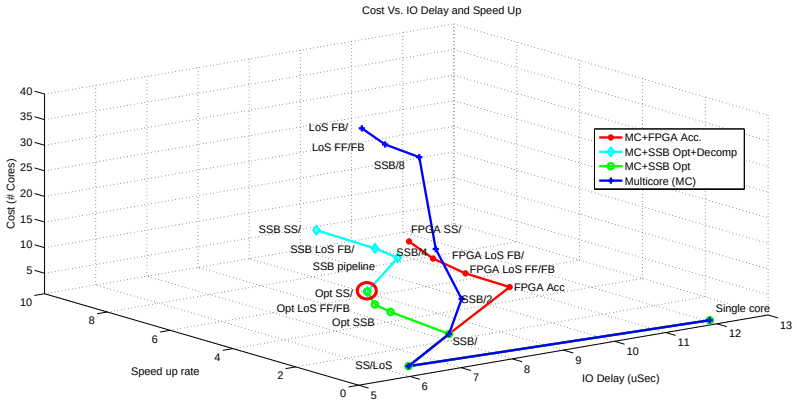


Figure 4.51: SSB optimization: cost Vs. IO delay and sampling frequency

4.2.7 Decomposition of the optimized SSB

Parallelization even more the application with the current data dependency may have a small impact in the application performance. Moreover, SSB is still the bottleneck of the application. As before, decomposition over the transformed SSB is proposed. Rewriting the Equations 4.5 and 4.6, this equations are obtained,

$$\mathbf{X}_{n+1} = \widehat{\mathbf{A}} * \mathbf{Pre}_n + \mathbf{B} * \mathbf{U}_n \tag{4.5}$$

$$\mathbf{Pre}_{n+1} = \widehat{\mathbf{C}} * \mathbf{X}_n + \mathbf{D}_{pre} * \mathbf{U}_n \tag{4.6}$$

Using both equations, the following decomposition for the SSB optimized is possible. This decomposition is represented in the Equations 4.7 and 4.8.

$$\begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\\ \mathbf{X}_3\\ \mathbf{X}_4\end{bmatrix}_{n+1} = \begin{bmatrix}\widehat{\mathbf{A}}_1\\ \widehat{\mathbf{A}}_2\\ \widehat{\mathbf{A}}_3\\ \widehat{\mathbf{A}}_4\end{bmatrix} * \begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\\ \mathbf{X}_3\\ \mathbf{X}_4\end{bmatrix}_n + \begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\\ \mathbf{B}_3\\ \mathbf{B}_4\end{bmatrix} * \mathbf{U}_n \tag{4.7}$$

$$\mathbf{Pre}_{n+1} = \begin{bmatrix}\widehat{\mathbf{C}}_1\\ \widehat{\mathbf{C}}_2\\ \widehat{\mathbf{C}}_3\\ \widehat{\mathbf{C}}_4\end{bmatrix} * \begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\\ \mathbf{X}_3\\ \mathbf{X}_4\end{bmatrix}_n + \mathbf{D}_{pre} * \mathbf{U}_n \tag{4.8}$$

These equations correspond only to the pre-sample task of the SSB. The resulting data flow graph can be seen in Figure 4.52

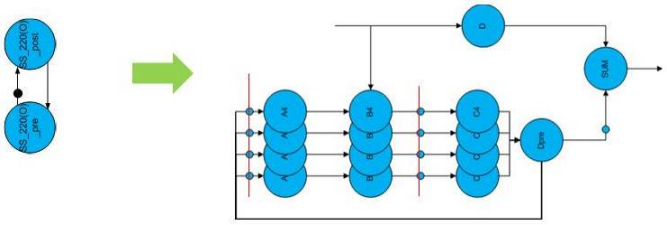


Figure 4.52: New data flow, optimized SSB

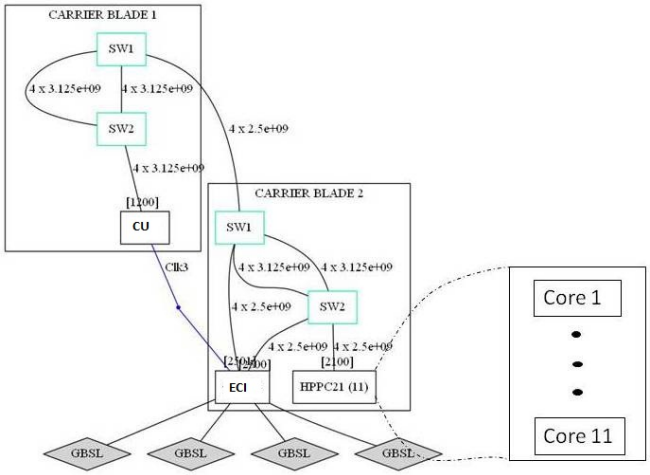


Figure 4.53: Multicore 11 cores Platform

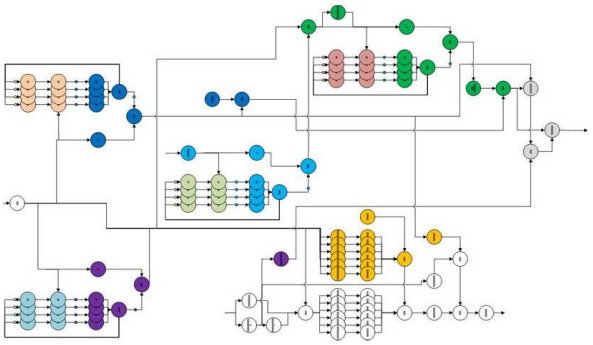


Figure 4.54: Data flow and mapping SSB pipeline

Pipelining

In Figure 4.52, two paths enclosed by delays are appreciated. Therefore, the application of pipelining is possible. To apply this strategy an additional resource (i.e., core) is added to execute one of the paths (e.g., $\widehat{A}\&B$). The resulting mapping and platform are depicted in Figure 4.53 and Figure 4.54.

Results for this model are shown in the Figure 4.55 and Table 4.16.

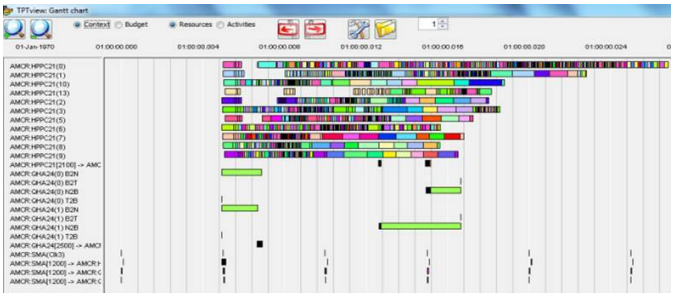


Figure 4.55: Scheduling results pipeline

Table 4.16: Performance results pipeline

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	44.047	46.097	47.835	7.4	8.7	9.82

As in previous cases, improving the SSB execution time by adding resources makes the LoS control loop the new bottleneck of the application. Therefore, parallelization is applied in the next step to reduce the overall execution time of this control loop.

Parallelization LoS FB control

Since the FB controller has six DoFs, the calculation of each degree can be executed concurrently. The mapping of Figure 4.57 is executed in the platform of Figure 4.56.

By applying this step, the sampling frequency greatly increases. The IO delay suffers again however with this step the speedup rises to almost 6 times. In the Gantt chart of Figure 4.58, it is clearly seen that the new bottleneck is the Core 1 in which the main tasks of the SS control loop are executed.

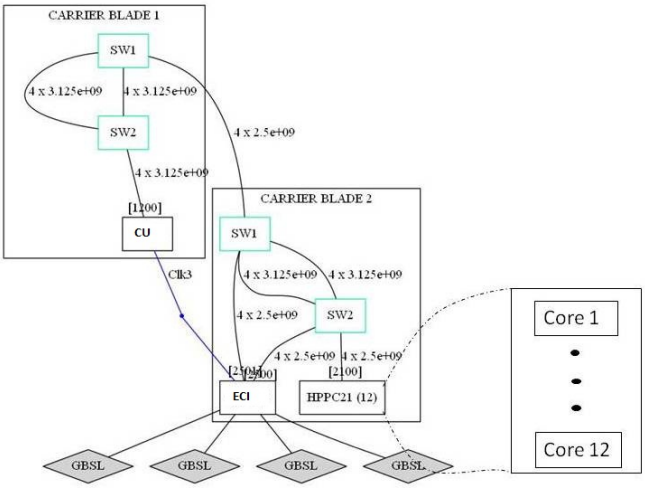


Figure 4.56: Multicore 12 cores Platform

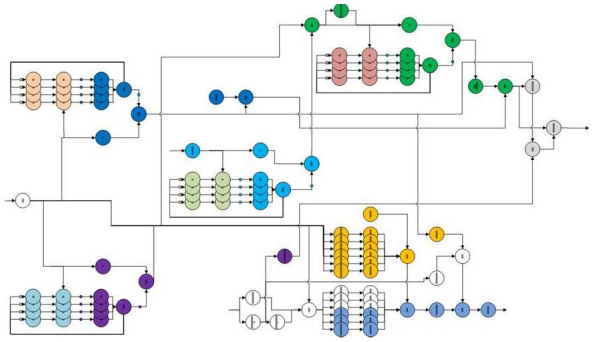


Figure 4.57: Data flow and mapping SSB Opt. LoS FB

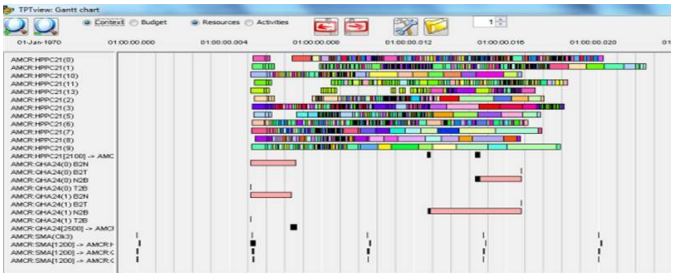


Figure 4.58: Scheduling results, SSB opt and LoS FB

Table 4.17: Performance results SBB Opt, LoS FB

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	48.486	53.470	58.609	7.66	8.66	10.22

Parallelization Short stroke

One new resource is added to the platform as shown in Figure 4.59. For the mapping the intention is to use the described mapping depicted in Figure 4.60. So far, half of the tasks of the SSB(FB) are deployed on core 1 with the SS. In this step, most of those tasks are mapped to a different core, see Figure 4.60.

The resulting performance is shown in Figure 4.61 and Table 4.18. By applying these changes, the IO delay performance and the sampling frequency performance are improved. The speedup obtained in this step is more than seven times the current motion control.

The result illustrated in the Gantt chart shows that the load is mostly equality distributed among the cores.

Table 4.18: Performance results SSB opt. Parallelization of SS

	Sampling Frequency (kHz)			**IO Delay (μs)**		
Samples	**Min.**	**Avg.**	**Max.**	**Min.**	**Avg.**	**Max.**
100	65.481	69.298	74.063	7.69	8.43	9.51

In Figure 4.62, the design-space is presented highlighting the position of this final step.

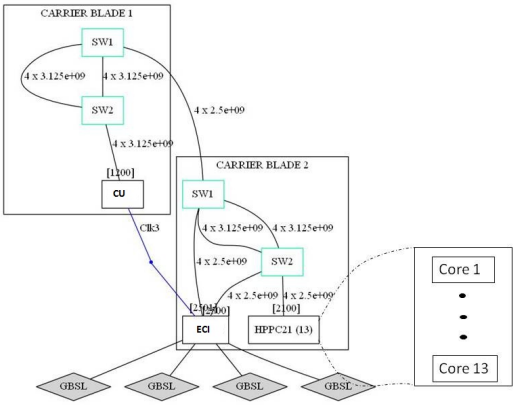


Figure 4.59: Multicore 13 cores platform

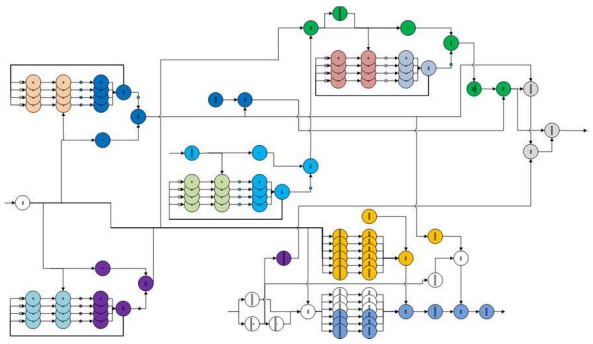


Figure 4.60: Data flow and mapping SSB opt. parallelization of SS

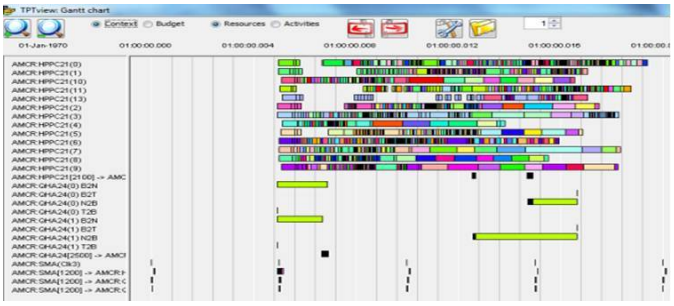


Figure 4.61: Scheduling Results, SSB opt and parallelization of SS

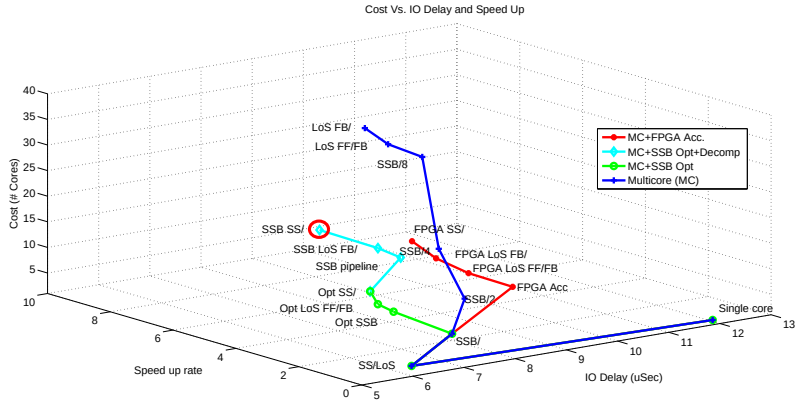


Figure 4.62: SSB optimization and decomposition: cost Vs. IO delay and sampling frequency

4.3 Solutions Comparison

The design-space exploration has permitted to build strategies to increase timing performance. In this section, a compilation of the best solution for each strategy is presented.

To give a more detailed comparison, two additional characteristics are introduced. Applicability refers to cases of the state space where the solution can be applied. This is a particular condition for the optimization solution presented in sections 4.2.6 and 4.2.7.

The second characteristic is Complexity. As the name says, it shows actions to take in order to implement a

strategy. The comparison of these metrics is given in Table 4.19. In the Table, it can be seen that the cost of a core and an FPGA has been discriminated.

Table 4.19: Dominant design solutions.

Solutions	**IO Delay (μs)**	**Sampling Frequency (kHz)**	**Cost**		**Applicability**	**Complexity**
			Cores	FPGA		
Single Core	10.86	3.77	1	0	General	Current Situation
Multicore & Decomposition	7.35	57.44	36	0	General	Off-line block decomposition
Multicore & FPGA Acc.	9.37	67.1	5	1	General	Accelerator integration
Multicore & SSB Opt.	6.81	46.87	7	0	Only if C is invertible	Off-line block transformation
Multicore & SSB Opt. & Decomp.	9.51	74.06	13	0	Only if C is invertible	Off-line block transformation and decomposition

All multicore solutions executes faster than 10 kHz sampling frequency. In contrast, neither of the solution could match the results obtained in [4]. The reason is that in the FPGA has finer granularity permitting to exploit even more the concurrency of the application.

All solutions presented have advantages and disadvantages over other solutions. Among these strategies which are applicable to general cases, The "Multicore & Decomposition" strategy shows the best IO delay performance result. But, the cost is huge compared to other solutions.

The "Multicore & FPGA Acceleration" is the second strategy in the general case group. The sampling frequency is one of the highest with really a low cost. Though, the complexity of implementing this strategy lies with the integration with the processor. If this is unrealistic, the obtained results would be further away from the reality.

The "Multicore & SSB optimization" gives an alternative to easily reduce the bottleneck in the application. Hence, the cost to achieve a higher frequency and a small IO delay is low. The drawback is that this technique is not applicable to all cases.

The addition of decomposition to the previous strategy increases the complexity. However, it results in increasing the sampling frequency considerable without much additional cost. The penalty is a higher IO delay.

Chapter 5

Conclusions

Given a benchmark application of a complex motion control, this project performs a design-space exploration to answer the research question about the sampling frequency and the IO delay that can be obtained on a multicore platform. This chapter presents the final conclusions, discussion, recommendation and future work.

5.1 Discussion

The design-space explored in this project is shown in Figure 5.1. This design-space has mainly three parameters: IO delay, speedup and cost. The IO delay is the time that the system takes to respond to an input signal. The speedup is the rate between the sampling frequency of the design and the current application sampling frequency, which is *10 kHz*. Ten times speedup corresponds to *100 kHz*. The cost refers to the number of cores plus the number of FPGA's used in the design. An FPGA is included since the design-space exploration considers hardware acceleration.

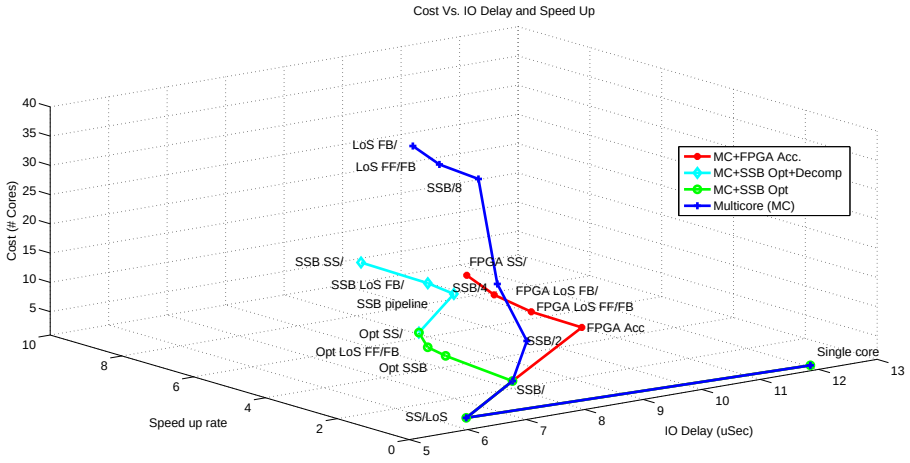


Figure 5.1: Multicore Design-Space Overview

The design-space exploration has as start point the single-core model. For this model a calibration step took place. This calibration step focused on a new flexible block added to the application since the measurement data of the rest of blocks are contained in the tool library. Some optimizations permitted to improve the initial measurement data of this block. At the end of this step, the measurement of this block was obtained and fed into the model.

Four strategies were used during the exploration to improve the performance of the application. The focus of these strategies was the heavy computational task of the system. These strategies are parallelization, decomposition, FPGA acceleration and an optimization strategy.

By adding applicability and complexity as metrics, design solutions can be compared. Following the parallelization and the decomposition technique, a solution with a high sampling frequency of *57 kHz* (or 5.7 speedup) and a good IO delay of 7.35 μs is obtained. However, the cost is high when compared with other solutions, see "Multicore (MC)" in Figure 5.1.

The parallelization strategy, combined with hardware acceleration strategies, easily permits to achieve a high sampling frequency of *67 kHz* (or 6.7 speedup) with a low cost of 5 cores and 1 FPGA. However, the IO delay is the worst when compared with previous results. The implementation of this design solutions lies in the possibility of integrating an FPGA into the processor. See "MC+FPGA Acc." in Figure 5.1.

The optimization strategy reduces the execution time of a heavy computation task. By combining this strategy with parallelization, a design solution with a small IO delay of *6.81*μs and a low cost of 7 cores is obtained. However, this strategy is only applicable to cases where C is invertible. By adding decomposition to this strategy, the IO delay becomes worse but the sampling frequency considerably improves. See "MC+SSB Opt" and "MC+SSB Opt+Decomp" in Figure 5.1.

These dominant points fairly give an answer to the research question proposed in this project.

5.2 Recommendation

This graduation project was developed as a joint effort of ASML and the Embedded Systems Institute. This project found optimal design solutions in the proposed design-space. By analyzing these solution points, this study recommends to further investigate the hardware acceleration technique and the combination of decomposition and block optimization. In both cases, the trade-off is promising and the complexity is affordable.

5.3 Future work

This project has proven the capabilities of CARM2G modeling tools. In this project a motion control application was studied. Motion control is one of many ASML systems. It could be interesting to study these systems using these tools as well.

From the point of view of the tool, it could be interesting to study how to properly support modeling of hybrid platforms. Strategies used in the design-space exploration were manually executed. It will be interesting to automatize these strategies. For example, parallelization could use an automatic tool that splits evenly the load among a certain number of cores. Similarly, an automatic decomposition tool could read the block specification and choose different techniques to split the task in smaller tasks.

Appendix A - Short Blocks

Abbreviation	Description
CO_SPG_FF	Feed forward control based on set-point differentiation.
CO_PID_LP	Pid feedback control.
AS_GS_6X6	Gain scheduling.
AS_LOS_DC	Long stroke specific disturbance correction.
MS_LOS2BM_COMBI	Long Stroke specific sensor transformations.
MS_LOS2BM_VER	Long Stroke specific sensor transformations.
MS_LOS2BF_COMBI	Long Stroke specific sensor transformations.
AS_SS_EMDC	(Non-linear) Electromagnetic disturbance compensation.
MAT_aXb	Multiply the input vector with a static matrix, eg used for gain balancing.
AS_SS_GS	Short stroke specific gain scheduling.
CO_VAR_GAIN	Position dependent gain using a linear gain boost and a dead zone.
SATURATION	Asymmetrical clipping.
FLT_X	The x^{th} order d-domain filter with smooth dynamic parameter change.

Appendix B - Abbreviations

Abbreviation	**Description**
LoS	Long Stroke.
SS	Short Stroke.
SSB	State Space Block.
FB	Feedback
FF	Feedforward.
GPP	General Purpose Processor.
FCFS	First Come First Serve.
FPGA	Field-Programmable Gate Array.
DoF	Degree of Freedom.

Bibliography

[1] *"ASML"*, ASML Technology, Wiki Page 2012.

[2] *"CARM 2G"*, ASML Internal Documentation, Wiki page 2012.

[3] EUV α Tool ABCD Control Implementation, Marc van de Wal. ASML Documentation.

[4] T. Kamp; Raymond Frijns; *"Heterogeneous Motion Control Processing Platform on FPGA"*; Master thesis Eindhoven University of Technology; Eindhoven the Netherlands, 2012.

[5] J. Lapalme, B. Theelen, N. Stoimenov, J. Voeten, L. Thiele, E. Aboulhamid; *"Y-Chart Based System Design: A Discussion on Approaches"*; In *Nouvelles approches pour la conception doutils CAO pour le domaine des systems embarques, Universite de Montreal*, Ph.D. thesis, 2009.

[6] O. Florescu, J. Voeten, J. Huang, and H. Corporaal. "Error Estimation in Model-Driven Development for Real-Time Software," in proceedings of FDL, 2004, pp.228-240.

[7] V. Vijayasankaran; "Performance Modeling of Level Sensor."; Master thesis Eindhoven Uniersity of Technology; Eindhoven, The Netherlands, 2012.

[8] V. Camelo; "Multi-core CPU Exploration for CARM Host in ASML Technology."; Master thesis Eindhoven Uniersity of Technology; Eindhoven, The Netherlands, 2012.

[9] *"Wings Performance Modeling Tools, WPMT"*, ASML Internal Documentation, Wiki page 2012.

[10] *"Wafer Stage"*, ASML Internal Documentation, Wiki page 2012.

[11] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K.E. Arzen, *"How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime"*, Control Systems, *IEEE*, vol. 23, no. 3, pp. 16-30, June 2003.

[12] P.J. Denning, The Locality Principle, *"Communications of the ACM"*, Volume 48, Issue 7, (2005), Pages 1924

[13] GCC optimization flags. GCC 4.7 MAN Document.

[14] Friendland, Bernard; *"Control System Design: Introduction to state space methods"*; Chapter 3, McGraw Hill, 1986.

[15] Keshab K. Parhi. *"VLSI Digital Signal Processing Systems, Design and Implementation"*. Wiley Inter-Science 1999.

[16] http://www.ics.ele.tue.nl/ lvbokhov/poosl/rotalumis/ Last time seen, 20th of November 2012.