Enhancing the performance of software-in-the-loop simulation of printer paper paths

Fan, Y.

Award date:
2013

Enhancing the Performance of Software-In-the-Loop Simulation of Printer Paper Paths

Master Thesis

Yue Fan 0786103
Master Program of Embedded Systems, TU/e

Host Company: Research & Development Department, Océ Technology.
Supervisor Océ: - Dr. Lou Somers
Supervisor TU/e: - Prof.dr.ir. Twan Basten

author(s)
Yue Fan

Venlo, 21 June 2013

version
1.1

number of pages
1 / 75

Abstract

Océ Technologies is a world leading company that develops office printing and copying systems, high speed production printers and wide format printing systems as well as related software. The development process of complex printers and copiers includes multiple disciplines, for example, mechanical engineering, electronic engineering and software engineering. The software engineering focuses on the development of Embedded Software (ESW) which controls the behaviors of printers and copiers.

It is very important to test the Embedded Software as early as possible. Since the testing of Embedded Software depends on the hardware, Océ has developed a Software-In-the-Loop (SIL) simulation environment to test the Embedded Software for paper transportation in the early stage when there is no prototype available. The Software-In-the-Loop simulation is more efficient for testing.

Normally, the testing should be done as fast as possible in the simulation. Unfortunately, the performance of Software-In-the-Loop simulation is not as expected. The simulation time is several times slower than the time taken by a real printer for some printer models. So this project focuses on the performance of Software-In-the-Loop simulation. It is envisioned to develop a variety of approaches to speed up the simulations by both software and hardware.

To achieve this goal, some analysis of Software-In-the-Loop simulation has been done to find out the bottleneck for performance. The Sheet Manager, which is responsible for updating sheets and recording all the states of sheets, is the bottleneck for performance. The algorithm of Sheet Manager has been made more efficient. The most important step is making Sheet Manager multi-threaded to take advantage of modern multi-core processors.

The results show that the throughput of the Software-In-the-Loop simulation, the number of sheets that can be simulated per second, can be improved by 10% to 120% by using these technologies with a four-core processor. And it has also been proven that there is no conflict with the next generation Software-In-the-Loop simulation.

Acknowledgments

I would like to express my sincere thanks to my supervisors, Prof.dr.ir. Twan Basten and Dr. Lou Somers. With their guidance and technical discussion throughout the project, I completed this graduation project successfully. They made great contributions and gave pivotal suggestions to my project in every stage. I would like to express my special thanks to Ben van Rens, who supported me with the implementation and was always ready to help me with problems. Without his support for implementation, it would have become very hard to implement my work successfully. Also thanks to Henri Hunnekens for the knowledge of SIL simulation. I would like to thank Dr. Sander Stuijk for technical discussions, which broadened my horizons.

Thanks to Océ Technologies for providing me the graduation project in an open and flexible environment.

Table of content

Glossary

Argus	Argus is the visualization tool to show the animation of a virtual printer. It also has a command interface to do some operations to the virtual printer.
Duplex	Double-sided printing.
ESW	Embedded Software, which controls the behaviours of the complex system of printer.
Main Node	Supervisory control that controls several sub nodes.
Printing job	Printing job is a test case for the virtual printer to test functional or non-functional requirements. A simple and typical printing job is to print a certain number of sheets.
Rose RT	IBM Rational Rose RealTime. A development and code generation tool for real-time model-driven design.
SheetLogic	SheetLogic is the name of the paper path plant model used in SIL.
Sheet Manager	Sheet Manager is manager of sheets. It can update the status of sheets and record all the states of sheets.
SIL	Software-In-the-Loop simulation environment.
Simplex	One-sided printing.
Sub Node	Controller that controls I/O-devices and is controlled by a main node.
X1	A model of a big printer.
X2	The next generation of X1, which is more complex.

1 Introduction

Océ Technologies is a world leading company that develops office printing and copying systems, high speed production printers and wide format printing systems as well as related software. It was founded in 1877 and it has research and development facilities in Venlo where the prototypes for new products are being developed.

At a high level of abstraction, a printer has a paper path that handles the paper (or other substrates) flowing through the printer, a data path that processes the data being printed, and an ink path that manages the flow of ink used to print the images on the papers. This project focuses on the paper path. Figure 1.1 shows an example of a paper path layout. It contains pinches, heaters, segments as well as sensors for the sheets; it receives sheets of paper from a Paper Input Module (PIM) and sends paper to a finisher (FIN). The development of printer paper paths involves designers and engineers from different disciplines, such as mechanics, electronics and software. As the disciplines are not equally involved simultaneously, problems may occur in later phases of the project. As the embedded software is expected to be developed and tested at the beginning, Océ is using a simulation-based testing approach, called Software-In-the-Loop (SIL) simulation.

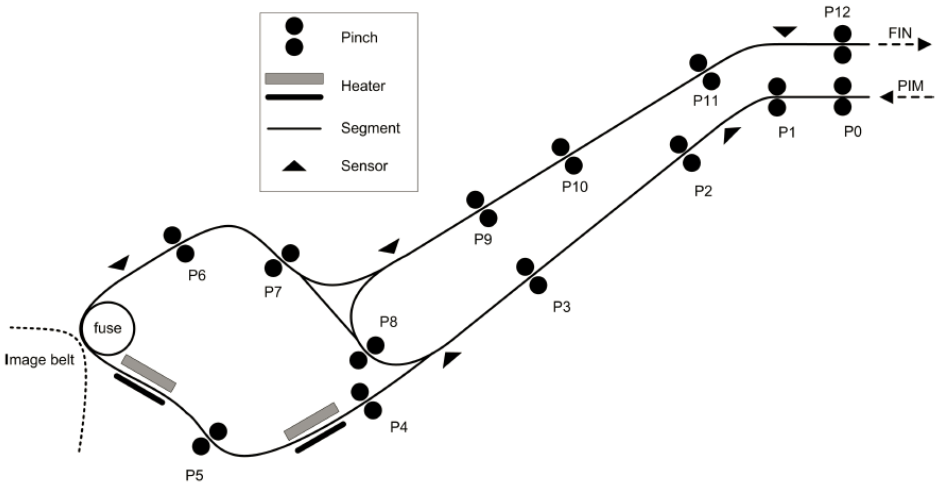

Figure 1.1 (from [2]): The layout of a paper path.

There are three main components in the high-level structure of the simulation environment (see Figure 1.2). The block SheetLogic simulates the hardware of a printer paper path, for example, sensors, actuators and also the paper flow. The block EmbeddedControl contains the embedded software. The third block, Animation is used to visualize the movement of the sheets and the states of hardware. Part of the embedded software is included in the block SheetLogic, such as the low-level control simulation, because it is dependent on the hardware.

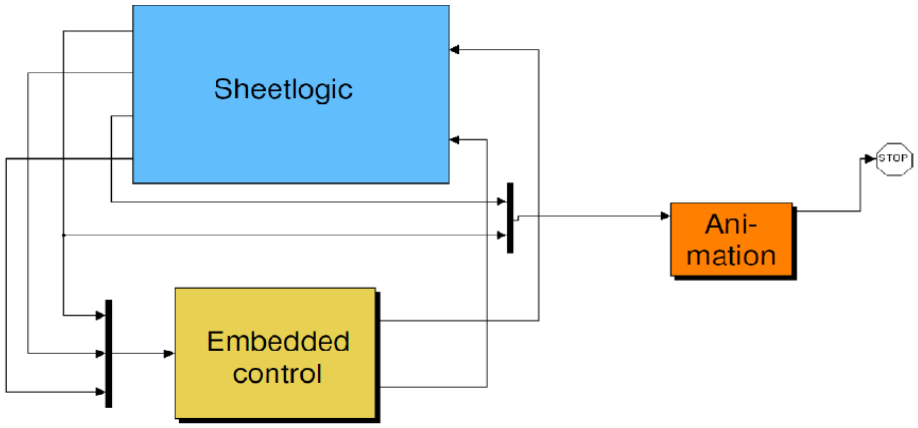

Figure 1.2 (from [2]): High-level structure of the simulation environment.

Unfortunately the performance of SIL is not as expected. The simulation time for a task is several times bigger than the time for that task taken by a real printer for some models. It is envisioned to develop a variety of approaches to speed up the simulations by both software and hardware.

The Master project focuses on the performance of SIL simulation. The purpose of this project is to enhance the performance of the SheetLogic in SIL simulation by applying a smarter algorithm to the active sheets and making the Sheet Manager parallel. The Sheet Manager is responsible for all the sheets. Chapter 2 introduces the SIL simulation environment. Chapter 3 focuses on the internal structure of SheetLogic and the profiling results of SheetLogic. Chapter 4 introduces the smarter algorithm for active sheets in the paper path. Chapter 5 presents the preparation of the multi-threading implementations. The design decisions of multi-threading implementation of the Sheet Manager are introduced in Chapter 6. Chapter 7 discusses the performance results. Chapter 8 pays attention to some case studies which are relevant for SIL simulation. In Chapter 9, some suggestions about future work are listed.

2 SIL Simulation Environment

This chapter presents the implementation of the SIL simulation environment to give a technical context for the project. The SIL simulation tooling has undergone several extensions and improvements since the first release in 2006.

Firstly an overview of the global structure is given. After that the structure of the SIL core is presented. Then the process of printing jobs and the model we analyzed is introduced.

2.1 Global Structure of SIL

The global structure of SIL did not change significantly over the time SIL was used. Figure 2.1 shows the entities and the relationships between them. Subsequently, several main entities are introduced.

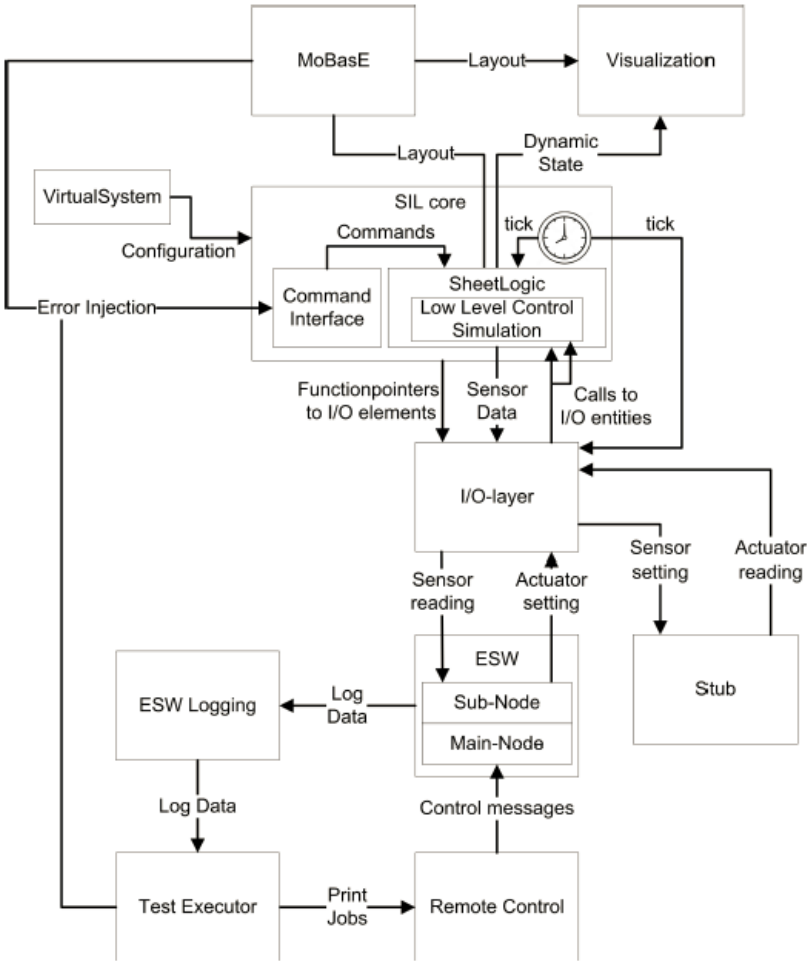

Figure 2.1 (from [2]): The current structure of the simulation environment.

SIL core and SheetLogic

The SIL simulation environment is a custom made tool implemented in IBM Rational Rose @ RealTime using C++. In the current SIL simulation environment, the most important plant model of the paper path is called SheetLogic. SheetLogic consists of different I/O devices and mechanical parts that are simulated to emulate the paper path behavior. The main types of devices are motors, sensors, pinches and sheets. Part of the embedded software is also included in the SheetLogic as low level control because it depends on the hardware.

The virtual system block in Figure 2.1 can be used to configure a virtual printer. After the virtual printer has been initialized, SheetLogic will keep on running with the simulation clock at 500Hz. In every clock cycle, the embedded software communicates with the SIL core via I/O layer by variables passing.

I/O-Layer

I/O-Layer is used to let the ESW communicate with the (virtual) hardware in the simulation environment. In each clock cycle, ESW gets the records of sensors and sends the command instructions to some actuators.

ESW

The block ESW contains the software that has to be tested. In the current SIL implementation, all the tests suffer from the performance of SIL.

Remote Control

The Remote Control is a java application that emulates the printer controller. Most of the simulation tasks can be done under Remote Control, for example, setting/reading the states of the virtual printer and start a simulation task of printing.

MoBasE

MoBasE is short for Model Based Engineering. The MoBasE is a data model framework that enables engineers to define a minimalistic data model that spans over disciplines and helps keeping important design information in one place. In the context of SIL, MoBasE is used to read the layout of the paper path of a certain printer for visualization and for generation of the SheetLogic plant model. MoBasE can provide the structure files of a certain printer for visualization.

Visualization

The visualization front-end of SIL is called Argus. Argus provides different functionalities to visualize the state of SheetLogic. The properties of the different parts (sensors, actuators, pinches, etc) can be

visualized in a list or in a 3D environment. Argus also provides functionality to manipulate the simulation by, for example, changing the values of sensors or actuators or stopping sheets. This is done via the so called command interface. Visualization seriously degrades the performance of SIL simulation. The information needed by Argus comes from the low level hardware and sheets. SheetLogic has to update the information to the structure of low level hardware and sheets in every simulation clock cycle. Then Argus can fetch the data with a customized frequency which can be changed by the end user.

Variable Database

The variable database contains the current value of all variables that are communicated between simulation modules. It plays an important role in the simulation of SheetLogic. In every clock cycle, the SheetLogic should get the variables of actuators from the variable database at the beginning and write the variables of sensor states to the variable database in the end. The variable database is not shown in the global structure diagram since it is a passive class rather than a capsule (an active class) and it can only be used by capsules.

2.2 Software Architecture of the SIL core

Figure 2.2 shows the overview structure diagram of the SIL core. In each clock cycle, the SIL core should be scheduled to make sure the simulation can continue. Each block shown in the graph is an active class (capsule). These capsules are triggered by the signal *p_Clock* according to the frequency of the simulation clock and have already been mapped to different threads. When a capsule finishes, it will notify the subsystem as it has been “scheduled”. After all the capsules are scheduled, the SIL core is scheduled and the simulation clock will gain a step to the next schedule time. To ensure the simulation, all the capsules should be “scheduled” in every clock cycle. Since all capsules can run simultaneously without dependencies between each other, it can be concluded that the slowest capsule determines the performance of SIL simulation if the simulation runs on a processor with enough threads.

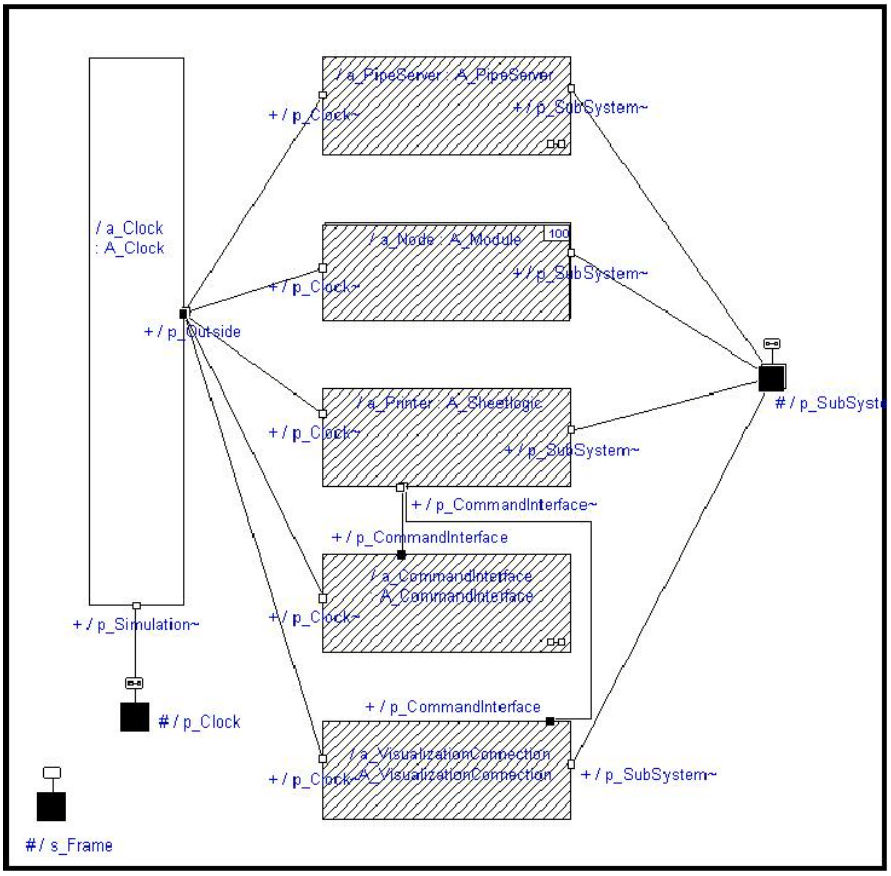

Figure 2.2: The overview of top level structure diagram of active capsules.

Since SheetLogic has the heaviest computation load among these 5 capsules, the performance bottleneck is apparently SheetLogic. To improve the performance of SIL simulation, SheetLogic should be optimized and then it is better to take advantage of modern multi-core processors to distribute the load.

2.3 Process of print jobs

To finish a print job, a virtual printer will experience several states through the printing process. Figure 2.3 shows the process of a print job.

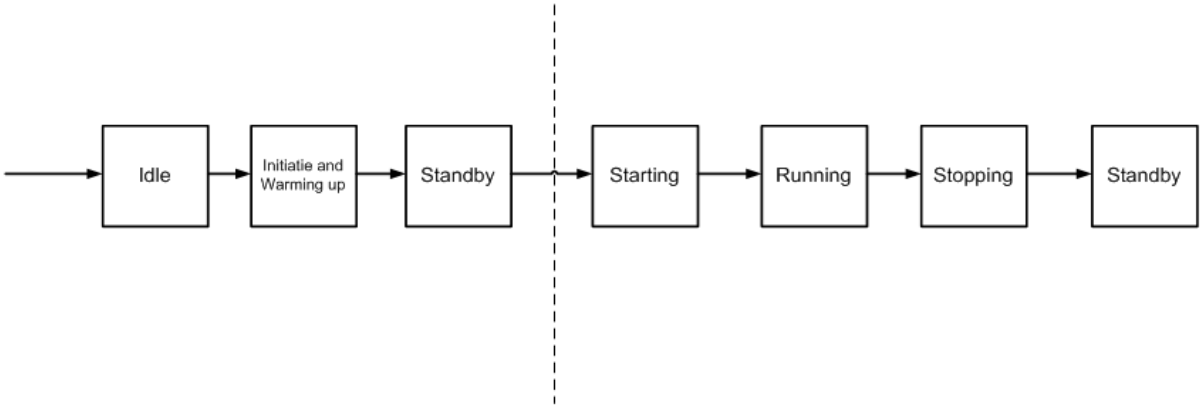

Figure 2.3: The process of a print job.

After the virtual printer starts, a print job can be assigned to the virtual printer. If the current state of the virtual printer is in “Standby”, it will start to process this job directly by entering the “Starting” state. Otherwise, the virtual printer will be forced to “Idle” before it turns to “Standby” state. The states before the dashed line in Figure 2.3 are less relevant in SIL simulation so that the following analysis will only focus on the performance after the virtual printer is already in “Standby”. For a normal print job, most of the simulation time is spent in “Running”.

2.4 Simulation model

As SIL only provides the simulation environment, a certain model of a printer which is going to be simulated is determined by MoBasE, which has been introduced in Section 2.1. The model used to analyze in this project is called X1. Some specifications of Model X1 which are relevant for SheetLogic are listed in Table 12.1 in the appendix.

2.5 Experiment setup

As introduced in Section 2.3, the virtual printer will be waiting in the “Standby” state. The simulation is done by scripting in the Remote Control. A timer is located in the remote controller. When the printing job comes, the timer will start to measure the simulation time. When the virtual printer finishes flushing and goes back to “Standby” state, the timer will also stop at this moment. The timer will use the clock of the operating system and the precision is one second.

Since the communication between Remote Control and SIL is on the same computer, the communication cost is very low. The simulation time result can be thought of as precise enough and repeatable when the system is not overloaded.

2.6 Performance of current SIL simulation

The performance of current SIL simulation is not satisfactory. It is slower than a real printer especially for huge printers. Table 2.1 shows the total simulation time for test cases when the virtual printer is ready in “Standby” for simulations, using the virtual printer X1. The results used in Chapter 2 and Chapter 3 are obtained by using Microsoft XP SP3 on Intel(R) Core(TM) DUO CPU E8300 @2.83GHz.

Simulation time (s)	Simplex	Duplex
100 sheets	162	273
300 sheets	357	632
500 sheets	543	1068

Table 2.1: Total simulation time (in seconds) of different printing jobs.

The simulation time consists of two parts: preparation time and printing time. The preparation time is to configure the virtual printer and flush the printer after printing has finished. It is determined by the model of the printer, the hardware which the simulation runs on, and simplex mode or duplex mode. The output delay which starts from the first sheet entering into the printer and ends when the first sheet gets out of the printer, can also be considered as a part of preparation. The output delay is fixed to the length of the paper path. Since the printing time for one sheet is very small, the total simulation time of a printing job of one sheet can be considered to be equal to the preparation time.

The preparation time is spent in state “Starting”, “Stopping”, part of “Running” and the transitions between these states. It is fixed to 68 seconds in simplex mode and 96 seconds in duplex mode on this PC. The other part, printing time, is about dealing with the sheets in the simulated print job, for example, moving and printing the sheets in the paper path. The time of this part increases linearly with the number of sheets. It is only spent in the “Running” state. Figure 2.4 shows the relationship between printing time and the number of sheets to be printed.

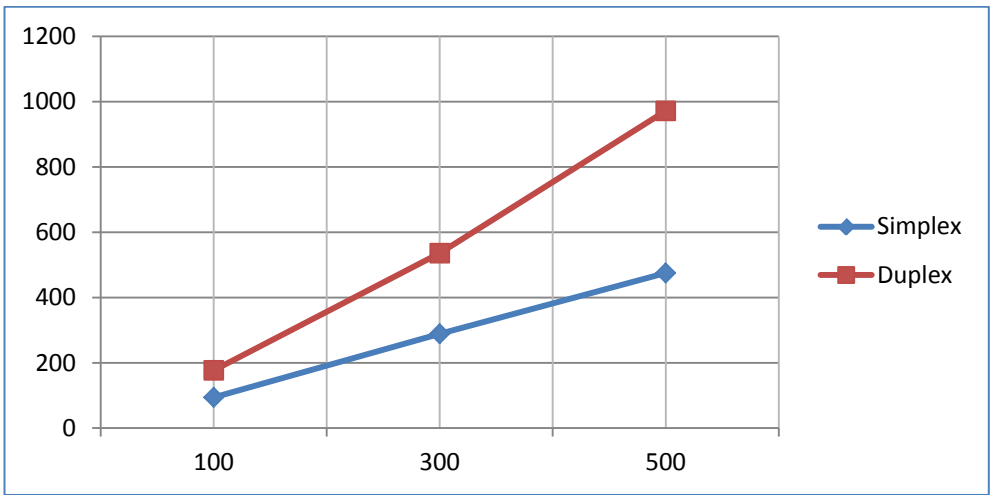

Figure 2.4 The simulation time for printing different numbers of sheets

3 SheetLogic description and analysis

The SheetLogic capsule, called A_SheetLogic in Figure 2.2, is used to configure the paper path hardware and update/record all the devices. When the simulation is running, SheetLogic communicates with the ESW via the I/O-Layer, simulation clock and variable database. Figure 3.1 shows the state diagram of the SheetLogic capsule.

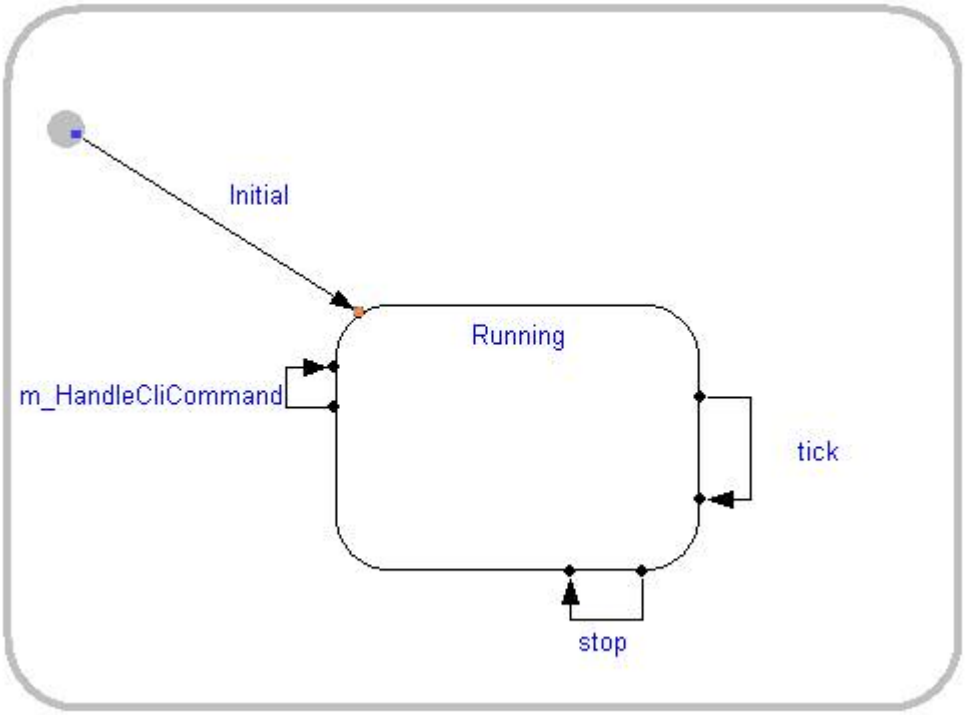

Figure 3.1: State diagram of SheetLogic.

3.1 Tick transition

In the Initial transition, SheetLogic reads the configuration settings and values and registers itself with the simulation clock. After this is done, it goes into the Running state and waits to be triggered by the simulation clock. The simulation clock ticks trigger the “tick” transition of SheetLogic at the frequency of 500Hz. The functions in the “tick” transition are listed below in Table 2 as subtasks. In the current simulation, these subtasks are executed sequentially in the order of their ID on a single thread within the SheetLogic capsule frame.

Task ID	Abbr.	Function Name	Description
1	Sleep	Sleep(v_delay)	Clock adjustment.
2	MA_upd	v_SheetLogicModuleAdapter.updateValues()	Update the variables in the structure from the Variable Database.
3	IO_Act	v_SheetLogicIOLayer.updateActuators()	Read from structure and update actuators
4	Upd_Act	Actuator::updateActuators()	Update actuators.
5	Upd_Sen	Sensor::updateSensors()	Update sensors.
6	Upd_Mtr	Motor::updateMotors()	Update motors, let motors calculate their new states.
7	Upd_Sht	D_SheetManager:: Instance()->updateSheets()	Update all the active sheets. The default number of active sheets is 150.
8	Upd_Tray	D_Tray::updateTrays()	Update trays (will be extended).
9	Upd_Poi	f_UpdatePois()	Update points of interest if there exist.
10	Upd_Dev	D_Device::updateDevices()	Update all the devices (will be extended).
11	Rec_Sen	Sensor::recordAllStates()	Record all the states of sensors.
12	Rec_Act	Actuator::recordAllStates()	Record all the states of actuators.
13	Rec_Mtr	Motor::recordAllStates()	Record all the states of motors.
14	Rec_Pin	D_Pinch::recordAllstates()	Record all the states of pinches.
15	Rec_Sht	D_SheetManager:: Instance()->recordAllStates()	Record all the states of active sheets.
16	IO_Sen	v_SheetLogicIOLayer.updateSensors()	Get sensor values and update structure.
17	MA_Rec	v_SheetLogicModuleAdapter.writeValues()	Write the variables to Variable Database.
18	Done	p_Clock.done().send()	Send the signal “scheduled”. Finish this transition and go to running state.

Table 3.1: Functions in the tick transition.

Updating is done by calculating the new states of hardware and sheets while recording means storing the information in the data structure or Variable Database. In one transition, when all the hardware and sheets have been updated, the internal clock of SheetLogic has finished synchronization with the simulation clock. Then all the states are recorded under the current simulation time. The SheetLogic has to communicate with Variable Database by a structure which acts as intermediate.

To speed up the tick transition, it is necessary to get the relationships such as dependencies between these sub tasks. Figure 3.2 shows the data dependencies and control dependencies between different sub tasks.

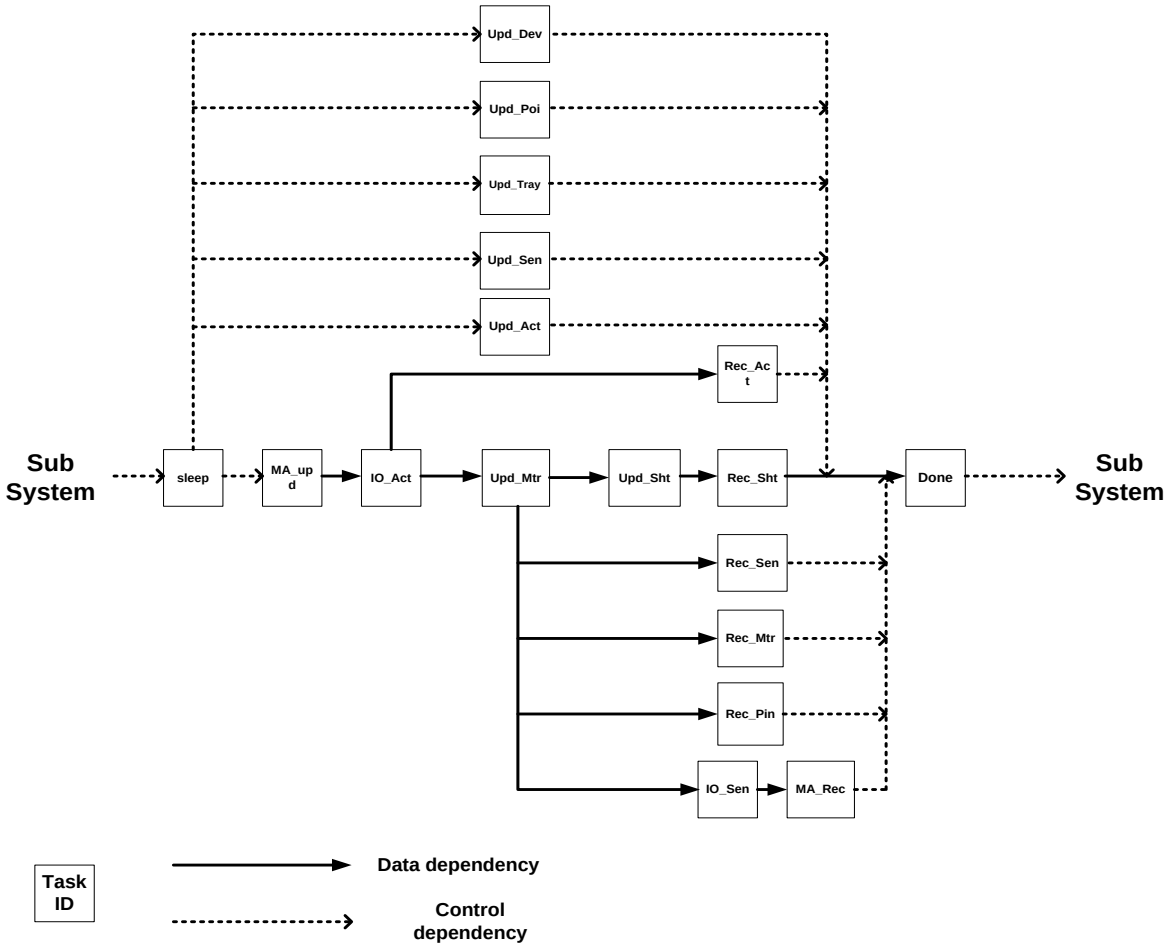

Figure 3.2: Dependencies diagram.

Performance is about timing. If there are enough cores that can be used to parallelize these tasks, the critical path would be the bottleneck of the whole transition. To get the critical path and the time it takes to execute once, profiling each sub task is necessary. The way to measure the execution time is shown in Appendix B.

For the tick transition, the one-shot execution time depends on the virtual printer's state. When the virtual printer is "Idle" or "Standby", there are no active sheets. The sheet manager has no active sheets to update and record. The execution time of task 7 and task 15 is then very short. Figure 3.3 shows the ratio of execution time of different tasks in "Idle" or "Standby" state. Table 3.2 lists the tasks which are the main computation load producers according to Figure 3.3. The execution time is obtained by using the timing service provided by Rational Rose Real Time. Section 12.2 introduces the timing service in Rational Rose Real Time.

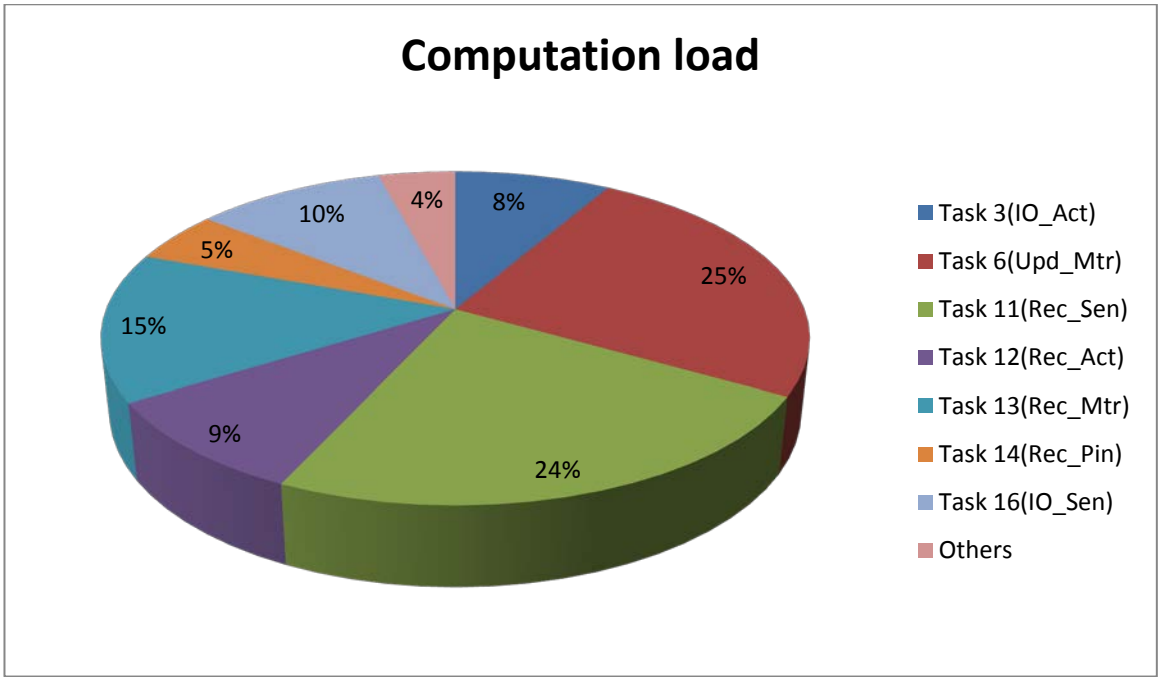

Figure 3.3: Ratio of computation load in “Idle” and “Standby”.

Task ID	Abbr.	Execution time in “Idle” and “Standby” (ms)
3	IO_Act	0,07192
6	Upd_Mtr	0,21250
11	Rec_Sen	0,20280
12	Rec_Act	0,07930
13	Rec_Mtr	0,12350
14	Rec_Pin	0,04255
16	IO_Sen	0,08810
Others	-	0,03551
Total	-	0,85613

Table 3.2: Main computation load producers in “Idle” and “Standby”.

In “Idle” and “Standby” state, the main computation producers are updating motors and record the states of all the hardware. Since there are no sheets in the virtual printer, only hardware updates and is recorded. In the current simulation, the virtual printer will spend some time under “Idle” and “Standby” before it is running.

While the virtual printer is in “Running” state, printing jobs are taking place. A sheet in the print job description becomes an active sheet when it enters the virtual printer. As more sheets come into the virtual printer, the list of active sheets increases. To limit the computation load caused by a large number of sheets, when the size of the list of active sheets is beyond the value set by the user (default value is

150), the sheet manager will remove some sheets which are already in the finisher to stay within the size of the list. Figure 3.4 shows the ratio of computation load in the “Running” state when the active sheets are saturated to 150. Table 3.3 lists the main computation load producer in the “Running” state according to Figure 3.4.

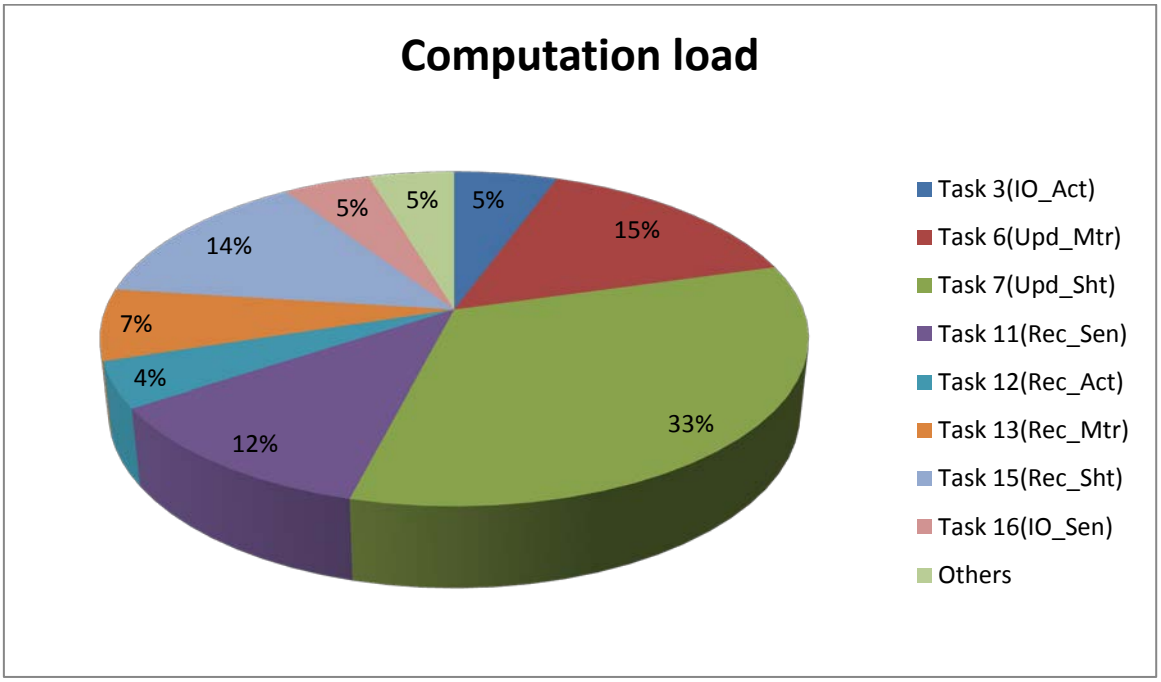

Figure 3.4: Ratio of computation load in “Running”.

Task ID	Abbr.	Execution time in the “Running” state (ms)
3	IO_Act	0,10467
6	Upd_Mtr	0,28234
7	Upd_Sht	0,62841
11	Rec_Sen	0,22040
12	Rec_Act	0,08226
13	Rec_Mtr	0,12991
15	Rec_Sht	0,25687
16	IO_Sen	0,08810
Others	-	0,08712
Total	-	1,88008

Table 3.3: Main computation load producers in “Running”.

Comparing the execution time of the tasks in different printing states, there is not too much difference for all tasks except for task 7 and task 15. Their load increases significantly when the state changes from “Idle” or “Standby” to “Running”. The increase of the execution time in the “Running” state compared to “Idle” and “Standby” is caused by the sheet manager, by updating and recording the active sheets. Task 7

(Upd_Sht) and task 15 (Rec_Sht) will produce 47% of the total load under “Running” when the active sheets have saturated to 150.

3.2 Sheet Manager

To have a close look at the active sheets, there are actually two kinds of sheets whose positions are different. Sheets in the paper path are being printed and moving through the paper path and sheets in the finisher have already been printed and would not move anymore. Sheets in the paper path must be traced by the sheet manager; otherwise the simulation of the print job would fail. Figure 3.5 shows the fluctuation of the number of different sheets, for example 500 pages are to be printed, in simplex mode.

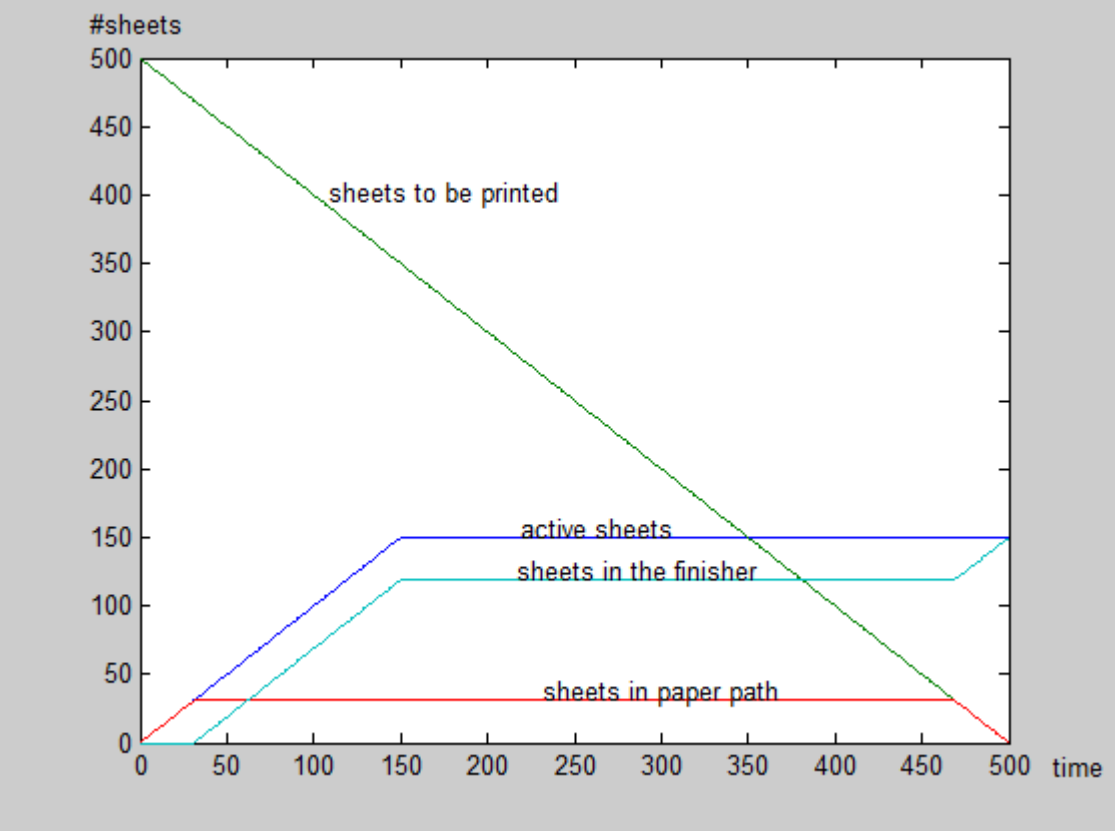

Figure 3.5: Fluctuation of the number of different types of sheets.

As sheets enter into the virtual printer, the number of sheets to be printed will reduce linearly after the first sheet enters the virtual printer. When the first sheet enters the virtual printer, it will become a sheet in the paper path. The number of sheets in the paper path will increase until the first sheet leaves the paper path. After the first sheet leaves the paper path, it will enter the finisher and become a sheet in the finisher. When the number of active sheets reaches the limitation, it would not increase any more. Some sheets in the finisher will be removed from the list of active sheets to keep the size of the list. After the last sheet enters into the virtual printer, the number of sheets in the paper path would decrease since no more sheets will come. The number of sheets in the finisher will increase. After all the sheets have been printed, there is no sheet in the paper path while all the active sheets are in the finisher.

The maximal number of sheets in the paper path is shown in Table 3.4. Since the paper path is longer in duplex mode, the paper path can contain more sheets.

Mode	Maximal number of sheets in the paper path
Simplex	28
Duplex	38

Table 3.4: Maximal number of sheets in the paper path.

It is obvious that the computation load would increase with the number of active sheets. The more active sheets, the more computation load. Figure 3.6 shows the relationship between the computation load of the sheet manager and the number of active sheets in simplex mode.

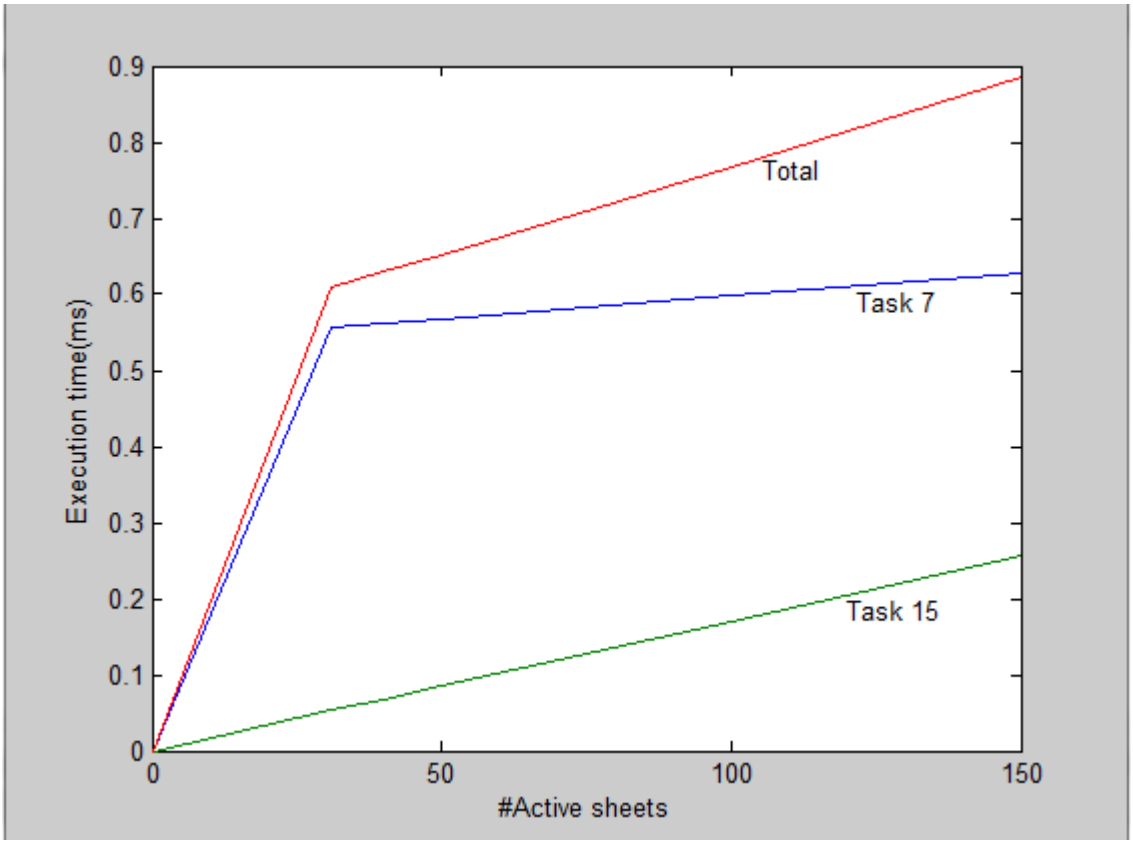

Figure 3.6: Execution time of Task 7 (Upd_Sht) and Task 15 (Rec_Sht) in simplex mode.

Task 7 Upd_Sht is used to update the sheets; it detects the pinches which are connected to the sheets and moves sheets through these pinches. Then it will notify sensors in the paper path to check whether this is done. If the sheet is already in the finisher, there is no pinch connected to it so that it cannot be moved. Then a lot of computation for movement can be skipped. This is why a sheet in the paper path has a much higher computation load for an update compared to a sheet in the finisher. The blue line in Figure 3.6 shows how the computation load increase slows down when the number of sheets in the paper

path saturates. Note that the number of sheets should be greater than or equal to the number of sheets in the paper path so that the virtual printer needs to update and record all the sheets in the paper path.

Task 15 Rec_Sht is used to record the states of all sheets. No matter where the sheet is, six kinds of states (leSpeed, lePos, LESegment, teSpeed, tePos, TESegment) need to be recorded in each clock cycle. So the computation load of this task is linear with the number of active sheets.

4 Algorithm improvement

As discussed in Section 3.2, actually there are two kinds of sheets in the list of Active Sheets: sheets in the paper path and sheets in the finisher. The main difference is about their positions. For a sheet in the finisher, it is not necessary to update or record it anymore because it cannot be moved and all the states stay the same. Updating and recording sheets in the finisher has three disadvantages:

1. It slows down the SIL simulation by doing something useless. Especially for Rec_Sht, it will record all the states of the sheets in the finisher in every simulation clock cycle although their states will never change.
2. It will destroy the load balancing for the multi-threading implementation of the Sheet Manager. Because the sheets will be distributed over different threads, it is better to have a balanced load for each thread. The computation load of sheets in the finisher is much less than that of sheets in the paper path. If we only update and record sheets in the paper path, the load will be balanced well naturally.
3. The visualization component uses the Active Sheets list; the sheets in this list are the ones shown in the visualization. If it is desired for some reason that more sheets should be shown in the visualization, the size of list of Active Sheets should be extended which will make the performance worse.

Since the list of Active Sheets is also used for visualization, it is necessary to create another list for sheets in the paper path if we want to improve performance. Then we only update and record the sheets in the list of sheets in the paper path. Because the sheets are moving, both of the lists will be updated in every simulation clock cycle.

1. The list of Active Sheets will be updated by the Sheet Manager. If the number of sheets in this list is beyond the default value, the oldest sheet in this list will be moved out from this list.
2. After Step 1, the Sheet Manager will go through the list of Active Sheets to generate a list of sheets in the paper path by the following rule. Only the list of sheets in the paper path will be updated and recorded.

The sheets in the finisher can be obtained with two conditions:

1. The sheet has no speed at all.
2. There is no pinch connected to this sheet.

These two conditions are necessary and sufficient conditions for sheets in the finisher. If a sheet has no speed and also does not connect to any pinches, it cannot be moved anymore so that it must be in the finisher. If a sheet is already in the finisher, it must have no speed and no pinch connected.

The new relationship between the execution time of updating and recording all the sheets and the size of the list of Active Sheets is shown in Figure 4.1.

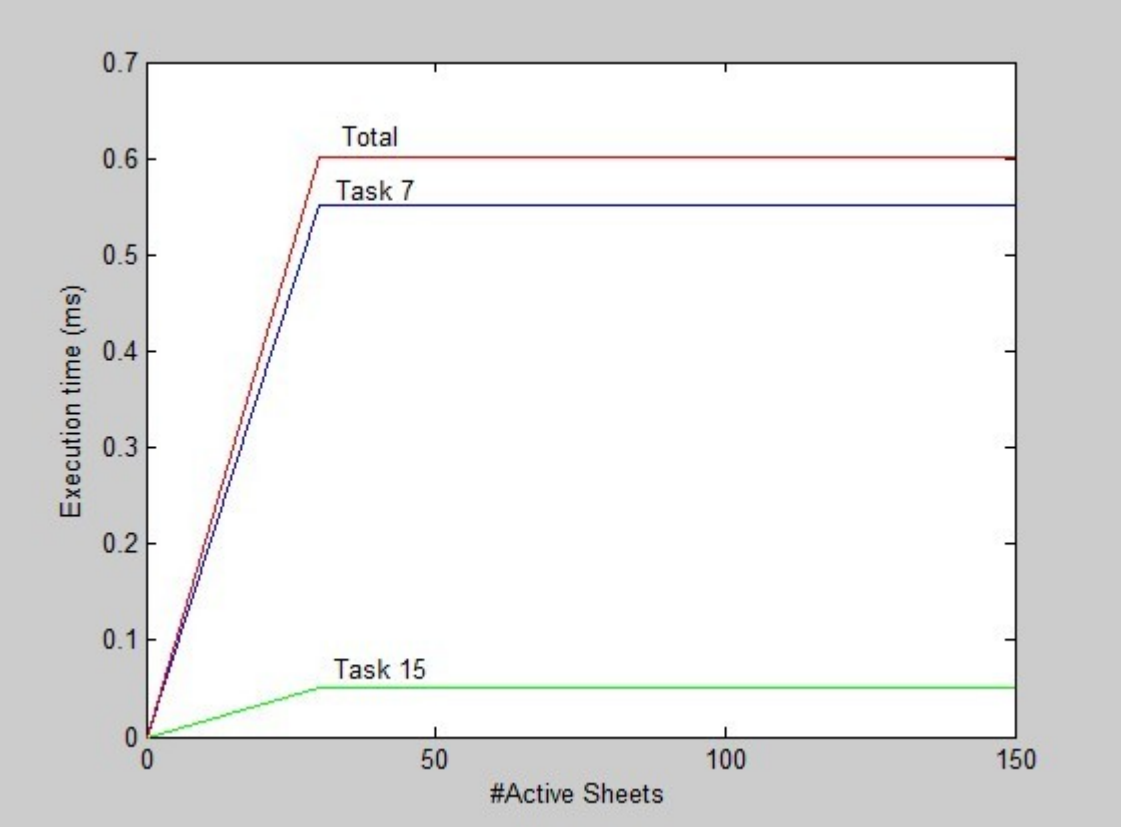

Figure 4.1: Execution time of Upd_Sht and Rec_Sht in simplex mode after separating the sheets.

From the figure, it is obvious that after the paper path is full, the execution time of both tasks will stay the same. For the default setting of the list of Active Sheets (the size is 150), the total execution time of these two tasks in one simulation clock cycle will decrease from 0.88525ms to 0.60189ms.

After the sheets in the list of Active Sheets have been separated, the three disadvantages listed before have been solved:

1. The performance has been improved. The Sheet Manager will not update and record sheets in the finisher.
2. All the sheets, which are updated and recorded, are in the paper path. The execution time of updating and recording any sheet in the paper path would be more or less the same. It is a very helpful preparation to balance the load for every thread in the future.
3. If the end-user wants to trace more sheets, he can extend the list of Active Sheets but the performance of simulation would be the same.

5 Preparation of multi-threading

This chapter presents the design decisions to prepare the Sheet Manager for multi-threading. In the first part, the architecture of parallelization has been discussed. The second part introduces the task merging for Sheet Manager while the third part gives the design decisions about threads creation. In the third part, how to create and use the threads efficiently is the important issue.

5.1 Architecture of parallelization

The most important approach to achieve speedup is to make SheetLogic run with multiple threads on a multi-core processor. Since the SIL will be restructured and the low-level control will be moved out from SheetLogic, the parallelization is applied to the sheet manager, in particular task 7 Upd_Sht and task 15 Rec_Sht. Figure 5.1 shows the process of task 7 Upd_Sht and task 15 Rec_Sht. The sheets come with a unique and incremental number.

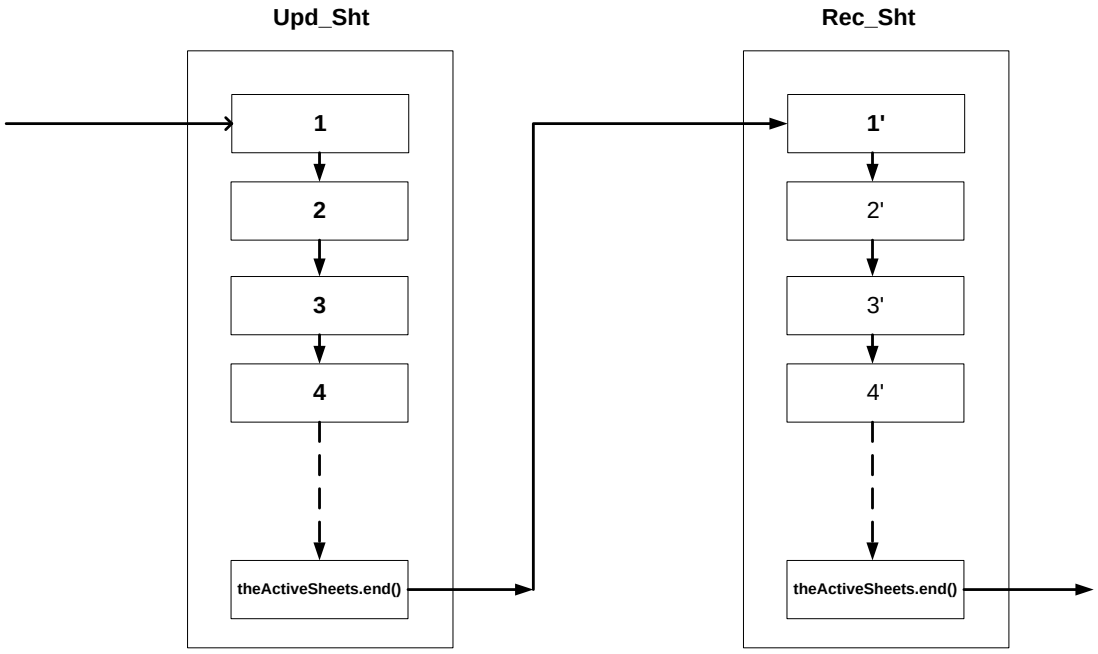

Figure 5.1: The process of updating and recording sheets in current SIL simulation.

The sheets are updated and recorded in sequential order. After all the sheets are updated in order, they will be recorded also in order immediately. This leads to a low efficiency. Actually, different sheets are independent from each other. This property can be used to parallelize the tasks.

5.1.1 Homogeneous solution:

This solution is to distribute sheets over different threads. All the threads will execute the same task for different sheets. The code should be duplicated. No communication between threads is needed and a one stop barrier should be set to make sure all the sheets are finished before next the task starts. Figure 5.2 shows the data flow of this possible solution assuming there are two free threads.

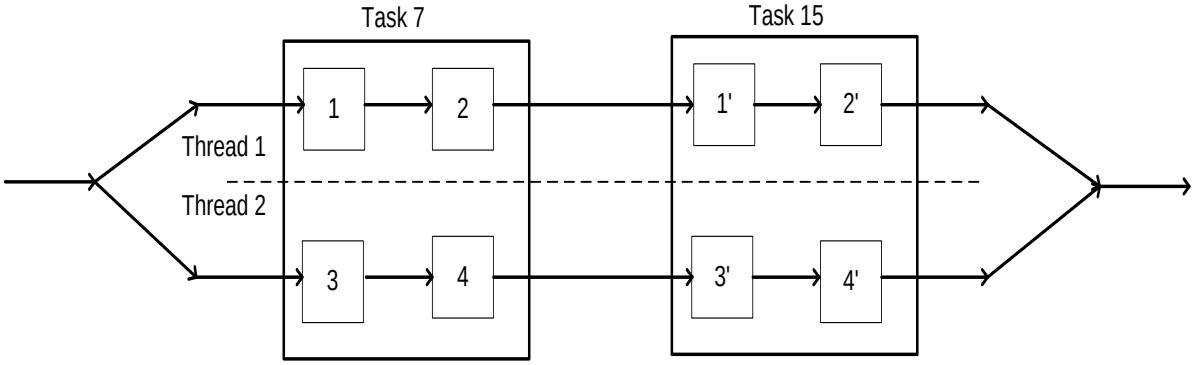

Figure 5.2: Flow diagram of the homogeneous multi-threading solution.

Assume the number of sheets is M and the number of threads is N, then at least $\left\lfloor \frac{M}{N} \right\rfloor$ sheets will be mapped to the same thread. Based on this solution, more optimizations can be made. The sheets in the paper path have more computation load than the sheets in the finisher. Most of this problem was already solved by the algorithm improvement, but it is still possible that sheets with different positions in the paper path have different computation load. For this case, the solution shown in Figure 5.2 is not optimal anymore.To solve this, the tasks should be distributed by module.

Furthermore, this design has to use some memories to store the global variables for recording the state between tasks. For example, sheet 1 has just been updated but the states should be kept until they have been recorded. If we record these states intermediately after the sheet has been updated, these global variables for states can be removed. Figure 5.3 presents the data flow diagram of homogeneous advanced solution.

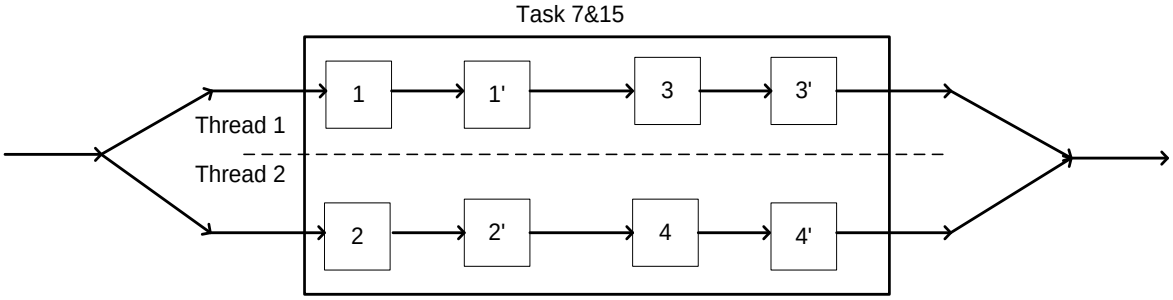

Figure 5.3: Flow diagram of the advanced homogeneous solution.

After merging task 7 Upd_Sht and task 15 Rec_Sht into one task, some global variables can be removed and memory usage will decrease. Another benefit is that load is well balanced no matter how much percentage of the sheets is in the paper path.

Although this advanced solution can improve the performance best, it is harmful to the structure and decreases the maintainability of SheetLogic.

5.1.2 Heterogeneous solution:

This solution is distributing tasks over different threads. Figure 5.4 shows how tasks are mapped onto different threads.

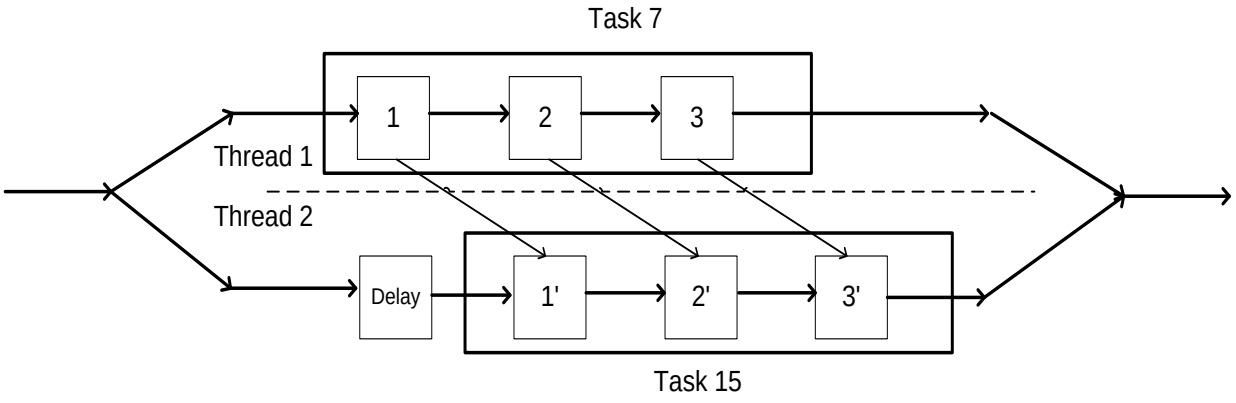

Figure 5.4 Flow diagram of the heterogeneous solution.

A sheet will be updated in task 7 and transfers its states to another thread which will record its states. After the first sheet has been updated, task 15 starts to record the first sheet. Since updating a sheet takes a longer time than recording a sheet, thread 2 will wait for thread 1 to pass the states. And the total execution time will be larger than the execution time of task 7 if the overhead of communication is taken into consideration. The load cannot be well balanced as well as it is not a scalable approach with more threads.

Table 5.1 compares the three possible solutions with several attributes.

Solution	Performance	Task decoupling	Communication	Load balance	Scalability
Homogeneous	Medium	Best	Intra-thread	Medium	Good
Homogeneous advanced	Best	Worst	None	Well-balanced	Good
Heterogeneous	Worst	Medium	Inter-thread	No balance	Bad

Table 5.1: Comparison between different solutions.

This project focuses on the performance of SIL simulation. The second most important factor is scalability since the SIL simulation will run on several computers with different processors in the future. The

homogeneous advanced solution is the best choice for performance. The heterogeneous solution is not suitable because the load cannot be balanced well so that the performance of the simulation would not gain so much. The homogeneous advanced solution will be implemented. But for X1, the load is balanced naturally, so the sheets will be distributed like the way in homogeneous solution, which is more simply.

5.2 Task merging

In every simulation clock cycle, two methods from the Sheet Manager are called in the tick transition of SheetLogic. They are Upd_Sht and Rec_Sht. The process of updating sheets and recording all the states of sheets is shown in Figure 5.1. The sheets are updated in order and after all sheets are updated, all the states of the sheets will be recorded.

In the improved SIL simulation, sheets will be distributed over different threads. In other words, every thread will update and record all the states of several sheets in every simulation clock cycle. The process of updating and recording in a thread is shown in Figure 5.5.

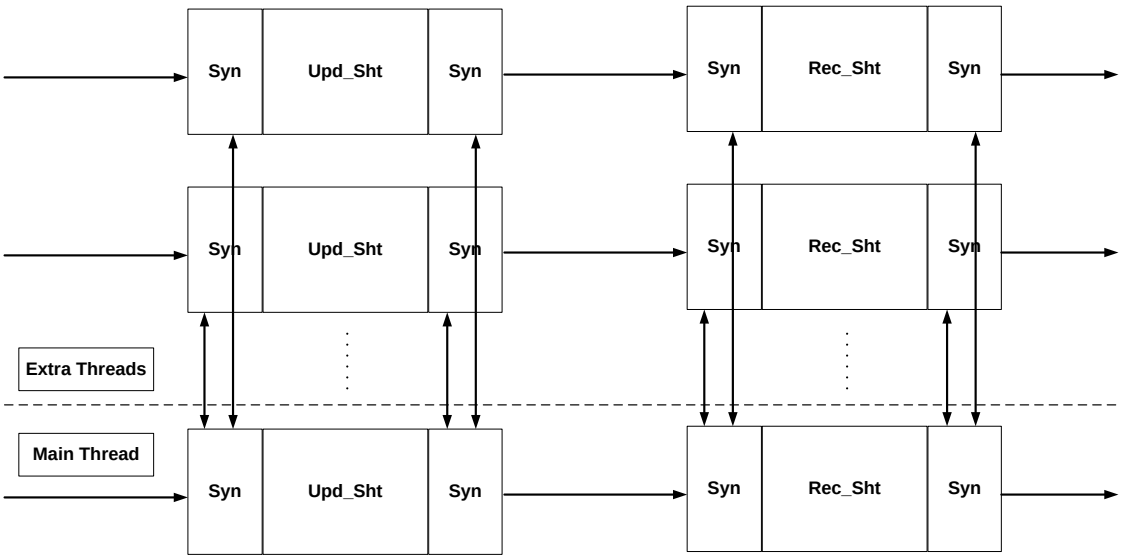

Figure 5.5: Task execution and synchronization in one simulation clock cycle.

It is necessary to synchronize the extra threads with the main thread. Before Upd_Sht and Rec_Sht, the extra threads need the information about which sheets will be updated or recorded. After Upd_Sht and Rec_Sht, the main thread needs the signal that all the extra threads have already finished their tasks. From Figure 5.2, it can be concluded that all the threads need 4 synchronizations in every simulation clock cycle. It is wise to decrease the number of synchronizations each time. Merging Upd_Sht and Rec_Sht can help to decrease the number of synchronizations from 4 to 2 in every simulation clock cycle. Figure 5.6 shows the process of updating and recording after merging them into one function.

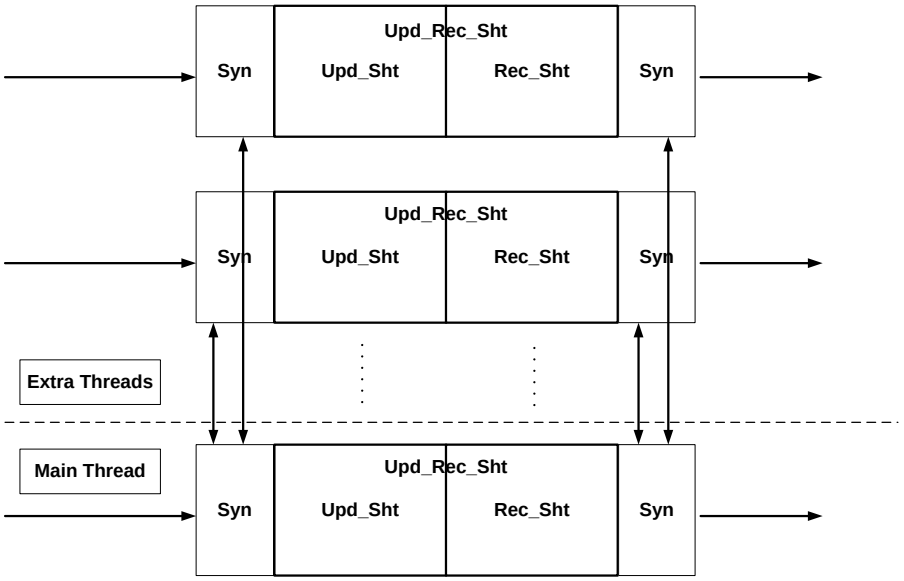

Figure 5.6: Task execution and synchronization in one simulation clock cycle after merging.

It would not improve any performance if the simulation runs in a single thread after merging. Task merging is only the preparation for a multi-threading implementation to decrease the time of synchronizations.

5.3 Thread creation

Thread creation is a very important issue in this assignment. For most implementations, programmers do not care when to create a new thread because the thread creation is cheap in contrast with the total execution time of this thread. However, in the SIL simulation, if threads are used to update and record sheets, they will be created and destroyed a large number of times. A sheet in the paper path is printed in about 6100 simulation clock cycles in simplex mode. And the number of sheets in the paper path is 28 in simplex mode which means a sheet is printed in about 217 simulation clock cycles in pipelining style. For example, if the virtual printer is used to print 500 sheets in simplex mode and in each simulation clock cycle only one extra thread will be created and destroyed, the number of thread creations and destructions would be more than 115,000. If more extra threads are created and destroyed in one simulation clock cycle, the number of thread creations and destructions would be more than $\#extrathreads * 115{,}000$. So, the overhead of thread creation and destruction must be taken into consideration.

The life cycle of an extra thread can be divided into three stages. The first stage is thread creation and parameters passing. In this stage, it will also spend some time to let the main thread check whether the extra thread is created successfully. In the second stage, the extra threads will execute some program and exchange parameters with other threads by using global variables. If the program has finished, the extra thread will be destroyed and checked in the last stage. Figure 5.7 shows the life cycle of an extra thread and its relationship with the main thread.

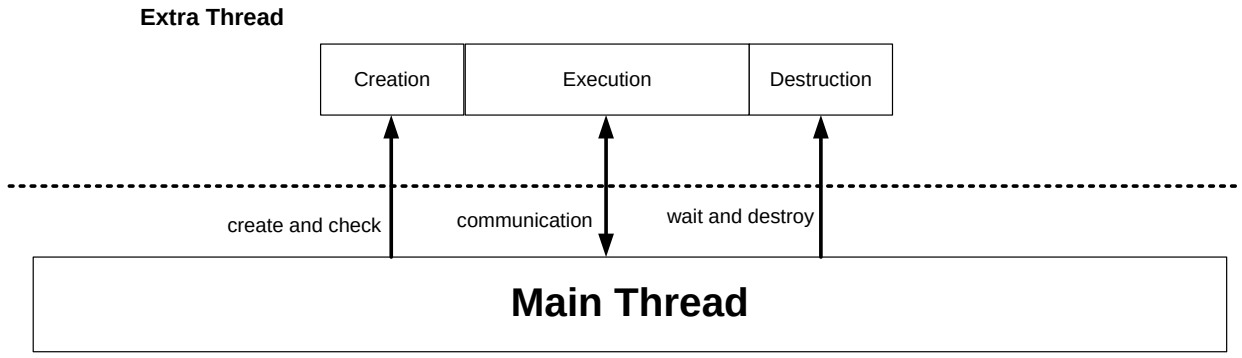

Figure 5.7: The life cycle of an extra thread and its relationship with the main thread.

The efficiency of an extra thread can be evaluated by the following equation:

$$E_{thread} = \frac{T_{exe}}{T_{crt} + T_{exe} + T_{des}}$$

T_{crt} and T_{des} is the time spent for creation and destruction which can be considered as the overhead of this thread.

To know the efficiency of extra threads in the SIL simulation, a small experiment is conducted to know the overhead of thread creation and destruction. The experiment is done in the following way:

1. Measure the execution time of updating one sheet by an extra thread which should be done in the extra thread.
2. Measure the total time of updating one sheet by an extra thread which should be done in the main thread.
3. The time obtained in Step 1 is T_{exe} while the time obtained in Step 2 is $T_{crt} + T_{exe} + T_{des}$.

The results of this experiment are shown in Table 5.2.

	Execution	Total time	Overhead	Efficiency
Time (ms)	0.029	0.112	0.083	25.9%

Table 5.2: The overhead and efficiency of using extra threads in one simulation clock cycle.

It is obvious that the efficiency of using extra threads to update one sheet is quite bad. The overhead is as big as three times the real execution. Is it possible to update and record several sheets by one extra thread which is created and destroyed in every simulation clock cycle? The basic idea is to increase T_{exe} but to keep the overhead the same. A new experiment is conducted in the following way to get the efficiency under the condition that the extra threads are created and destroyed in every simulation clock cycle.

1. Measure the execution time of updating half the sheets by an extra thread which should be done in the extra thread.
2. Measure the total time of updating half the sheets by an extra thread which should be done in the main thread.
3. For Step 1 and Step 2, the other half of the sheets are updated by the main thread and this experiment can only be done when the number of active sheets has already been saturated. When the number of sheets in the paper path equals the maximal number of sheets of this paper path, the experiment can start.
4. The time obtained in Step 1 is T_{exe} while the time obtained in Step 2 is $T_{crt} + T_{exe} + T_{des}$.

The results of this experiment are shown in Table 5.3.

	Execution	Total time	Overhead	Efficiency
Time (ms)	0.324	0.409	0.085	79.2%

Table 5.3: The highest efficiency of using extra threads in one simulation clock cycle.

If more extra threads are used, the efficiency will fall down with the number of extra threads. Figure 5.8 shows the relationship between the efficiency of each extra thread and the number of threads.

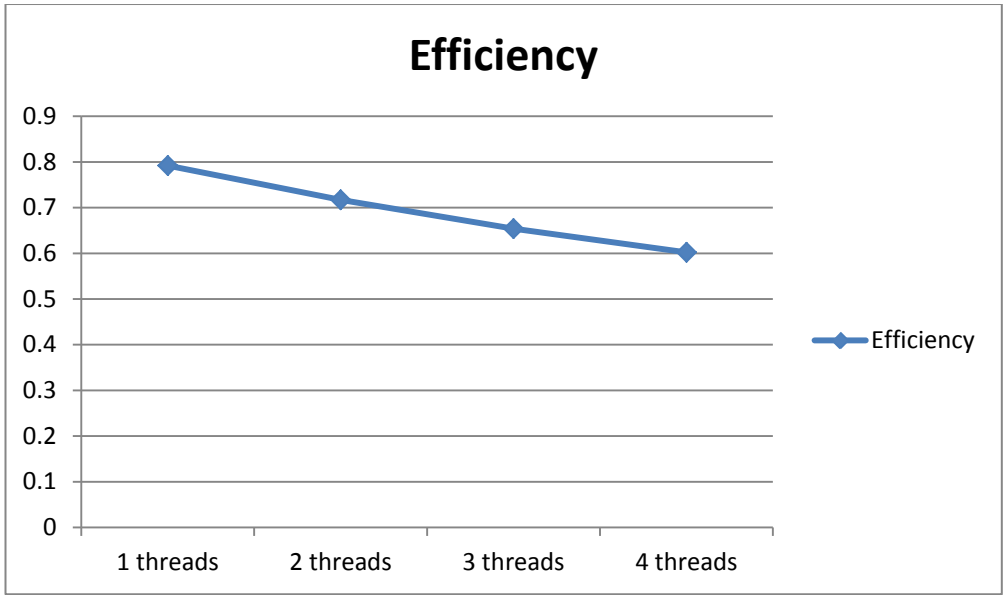

Figure 5.8: The efficiency becomes worse with the number of extra threads.

It can be concluded that the efficiency of extra threads is not satisfactory and it will become much worse if more extra threads are used. The bad performance and scalability gives too much overhead to the SIL simulation. To avoid the overhead, we should avoid creating and destroying extra threads multiple times. In other words, the extra threads should be created and destroyed only once so that these extra threads can be reused to improve the efficiency. In this way, we can avoid the overhead generated by thread

creation and destruction. Figure 5.9 shows the process of thread creation and destruction when done only once.

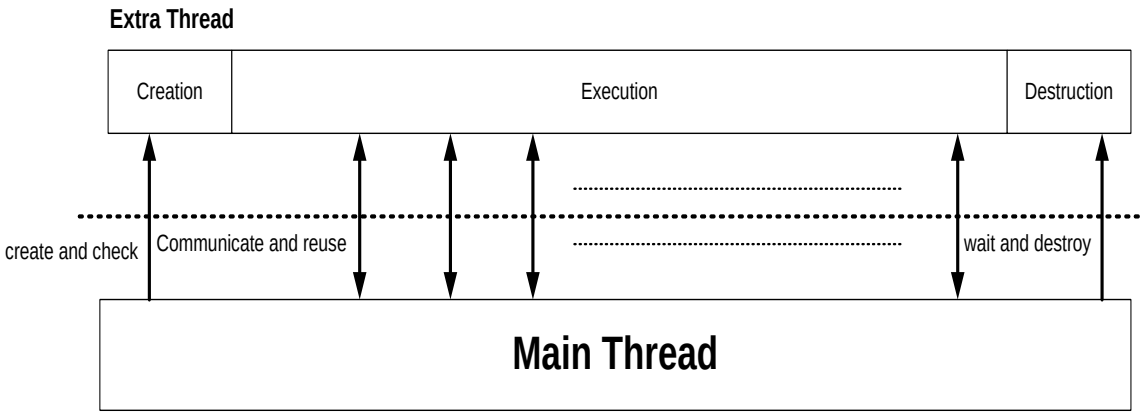

Figure 5.9: Extra thread creation and destruction for highest efficiency.

6 Multi-threading implementation

This chapter introduces and discusses the details of the multi-threading implementation for the Sheet Manager.

6.1 Time point of thread creation

There are two options for when to create extra threads:

1. Create extra threads when the Sheet Manager is being initialized.

   Advantage: There is no latency to simulate a printing job because the extra threads are already created. The time when the extra threads would be created can be determined so that the Sheet Manager does not need to check when the extra threads will be created.

   Disadvantage: The extra threads will consume CPU after they are created. Before a printing job comes, they will do nothing but busy waiting. They may interfere with other threads in the SIL simulation.

2. Create extra threads when the first sheet enters the virtual printer.

   Advantage: The Sheet Manager can create extra threads when it really needs them. They have little interference to other threads.

   Disadvantage: It takes some latency to create threads and check them. This amount of time will be added to the total simulation time. For small printing jobs, the latency is noticeable. And in every simulation clock cycle, the Sheet Manager has to check whether there are some printing jobs coming. This will also generate some overhead.

Decision:
The extra threads will be created when the Sheet Manager is being initialized. And just after they are created, they will be blocked immediately to avoid the interference with other threads. When they are needed, the main thread will release them. For the main thread, the latency of releasing a semaphore is much less than the latency for thread creation.

Rationale:
For the main thread, a semaphore can be considered as a simple variable. Releasing a semaphore is similar as changing the value of a variable. So, the execution time of releasing a semaphore is very short. If the main thread wants to create an extra thread, it has to be blocked until the extra thread has been created successfully. It can be concluded that thread creation has a much longer waiting time for the main thread than the semaphore releasing.

6.2 The scope of extra threads

There are two options for what is the scope of extra threads.

1. The extra threads are created and reused in the SheetLogic scope.

   Advantages: Other functions can also reuse these threads.

   Disadvantages: Currently, the SheetLogic depends on the Sheet Manager rather than the Sheets. If the threads to update and record sheets are reused by the SheetLogic, the dependencies from the Sheets to the SheetLogic must be added. However, the SheetLogic should not depend on Sheets when considering the software architecture. The dependency is not logically true. The pointers to different classes would make the parameter passing from the main thread to the extra threads more complex.

2. The extra threads are created and reused in the Sheet Manager. If other functions would use extra threads, they will create their own extra threads.

   Advantages: It will keep the software architecture the same as before. The type of pointers would only be unique.

   Disadvantages: The extra threads cannot be reused by other functions.

Decision:
The extra threads are created and reused only in the Sheet Manager.

6.3 Synchronization

As discussed in Section 5.2, there are two synchronizations between the main thread and extra threads in every simulation clock cycle. Two synchronization points can be set based on the dependency graph of the tick transition. Figure 6.1 shows the dependency diagram of functions in the tick transition with two synchronization points in red.

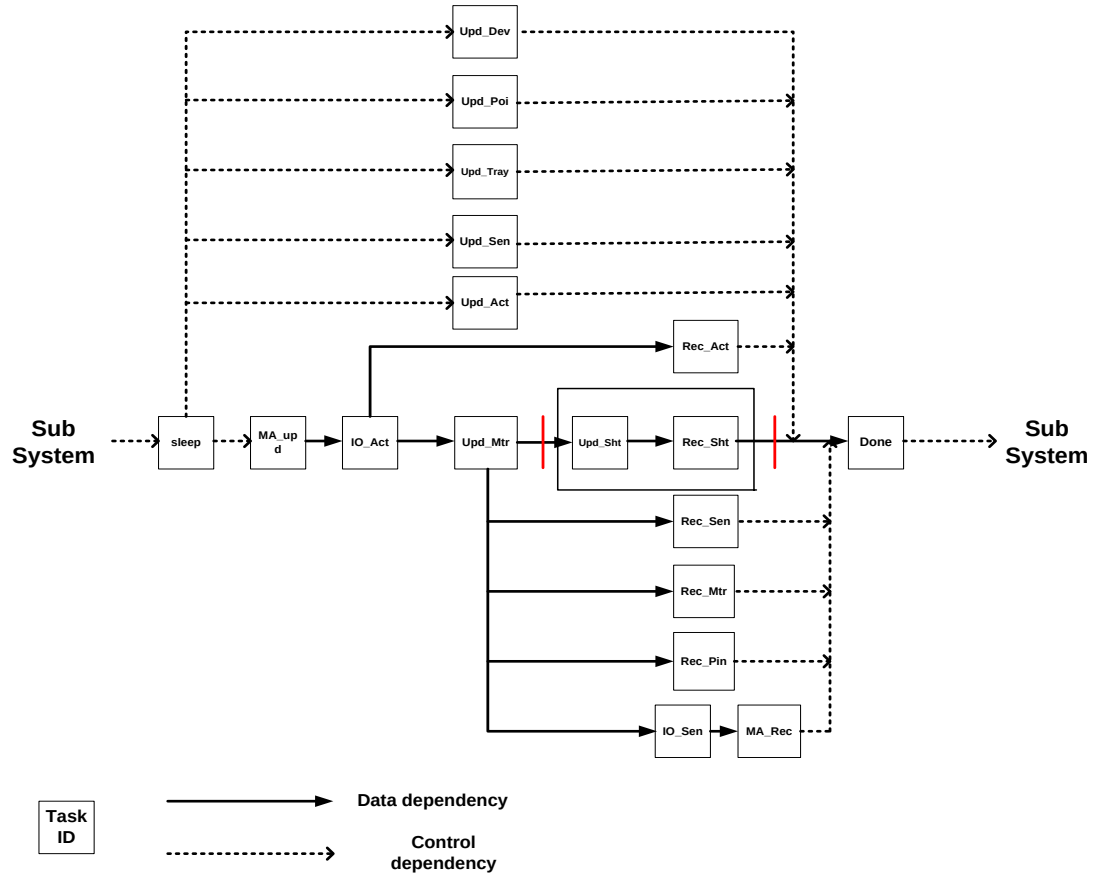

Figure 6.1: The dependency diagram of the tick transition with synchronization points in red.

To update sheets in the paper path, the Sheet Manager needs the current information of motors. The first synchronization point should be set after all the motors have been updated. The second synchronization point is a synchronization barrier before which every thread used by the Sheet Manager should finish its task. The position of the synchronization barrier is very important to the overall performance. Figure 6.2 shows the execution in one simulation clock cycle with a barrier set just after recording all the sheets. Here Rec_All_States is a combination of Rec_Sen, Rec_Mtr, Rec_Pin, IO_Sen and MA_Rec.

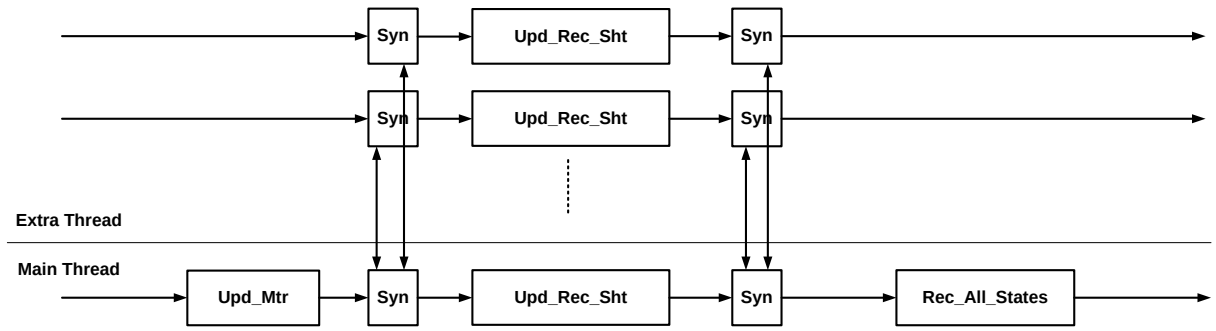

Figure 6.2: Barrier is set just after recording all the sheets.

The main thread and extra threads should finish Upd_Rec_Sht almost at the same time. In this case, the main thread does not pay too much time to wait for the extra threads. But there are around 200 threads running at the same time in SIL simulation, the extra threads are interfered a lot. This leads to them being much slower than the main thread. In every simulation clock cycle, the main thread has to wait for extra threads for a long time. To solve this problem, the barrier should be set as late as possible to hide the long waiting time for the main thread. The optimal position for the barrier would be after Rec_All_States as shown in Figure 6.3 since there is no dependency between Rec_All_States and Upd_Rec_Sht.

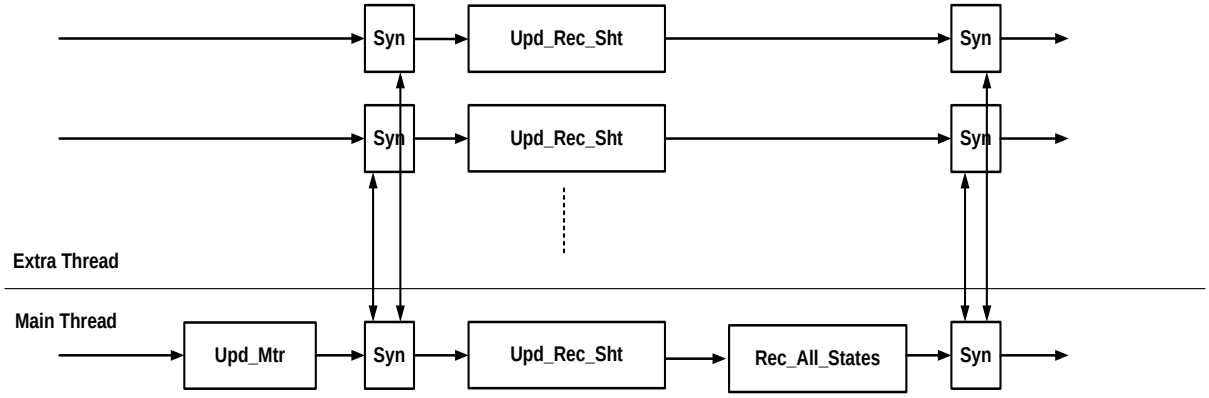

Figure 6.3: Barrier is set as late as possible.

The results of experiments show that the approach with a late barrier can decrease the total simulation time for printing 500 sheets in simplex mode by more than 5% with respect to.

6.4 Method of synchronization

6.4.1 Semaphores and spin locks

Since the extra threads are created at the beginning and will keep running until the virtual printer shuts down, they will execute a *while (true)* loop so that they can keep running. There are two options for the method of synchronization:

1. Using semaphores. In this way, the threads wait for the release of a semaphore; otherwise they will be blocked.

   Advantages: If threads finish their tasks, they will be blocked until the next release. It helps to decrease the average CPU usage and also the interference to other running threads.

   Disadvantages: Semaphores have overhead of sleeping, maintaining the ready queue and waking back up. It is not suitable for short period tasks. Figure 6.4 shows the semaphore mechanism. After the semaphore has been released, the blocked thread will spend some time to go through the ready queue and also wait for the processor. The time, which is spent in the ready queue, depends on the length of the queue. The sequence diagram of using semaphore is shown in Figure 6.5. Thread 1 is the master and thread 2 is the slave. Thread 2 will be blocked until thread 1 releases the semaphore. In SIL simulation, there are hundreds of threads. It will

increase the overhead of using semaphores because of the much longer waiting time in the ready queue.

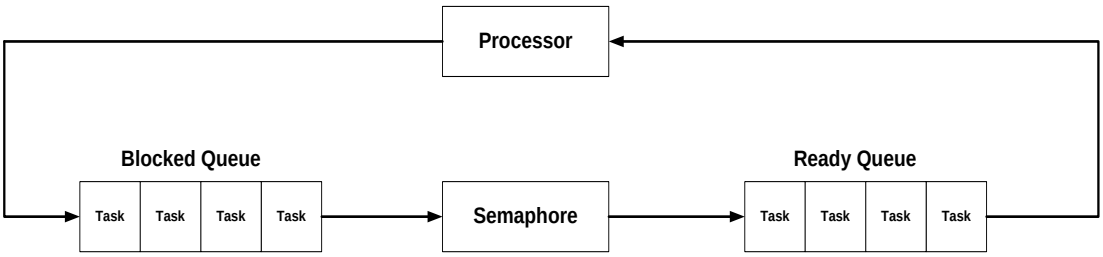

Figure 6.4: Semaphore mechanism.

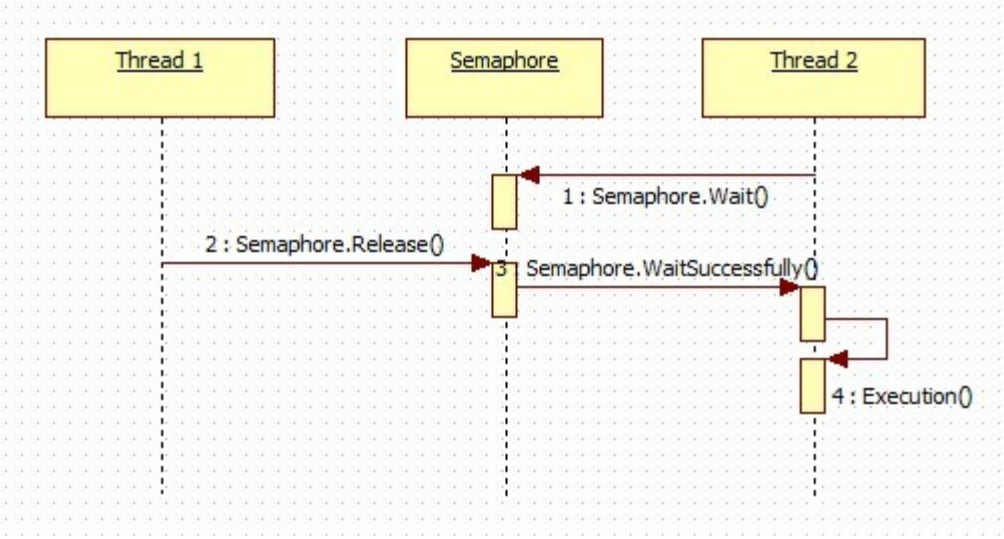

Figure 6.5: The sequence diagram of using semaphores.

2. Using spin locks (busy waiting). A spin lock is a lock which causes a thread trying to acquire it to simply wait in a loop while repeatedly checking if the lock is available. Since the thread remains active but is not performing a useful task, the use of a spin lock is a kind of busy waiting. Figure 6.6 is the sequence diagram of using spin locks. Before the value of the flag has been changed by Thread 1, Thread 2 will stay active to inspect the flag. As soon as the value of flag has been set as positive, Thread 2 will start to run with little delay because it is already active and occupying the CPU.

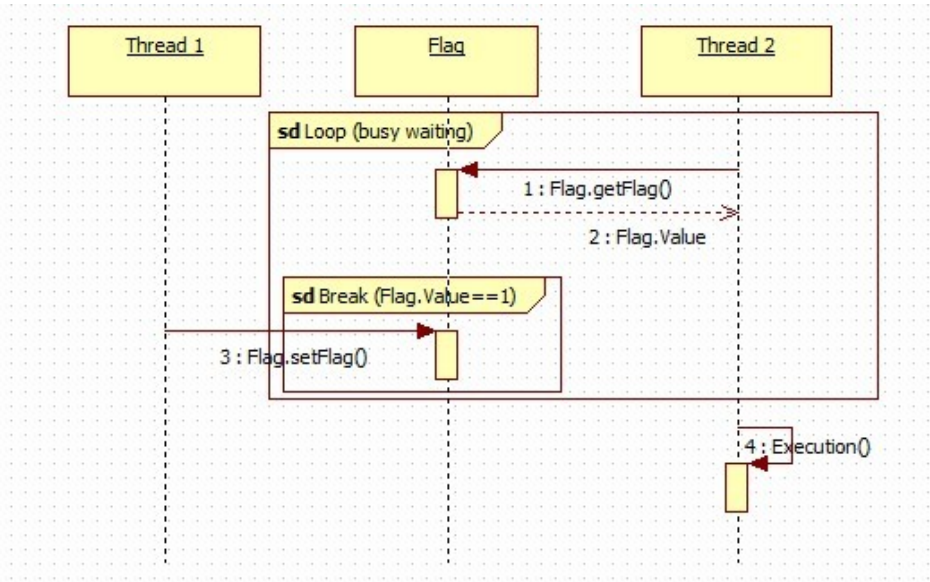

Figure 6.6: The sequence diagram of using spin locks.

Advantages: Spin locks can avoid the overhead from operating system process re-scheduling or context switching; spin locks are efficient if threads are only likely to be blocked for a short period.

Disadvantages: Spin lock will increase the average CPU usage significantly because it is a kind of busy waiting.

6.4.2 The second synchronization

Most of the overhead is the waiting time on the threads which are controlled by semaphores. If the main thread is waiting for more than once for releasing of extra threads, it will suffer a long waiting time. So for the second synchronization, the only possible method is to use spin locks.

6.4.3 The first synchronization

For the first synchronization, all the extra threads will be triggered by the main thread in every simulation cycle. There are three possible method synchronization ways for the first synchronization.

1. Semaphore. Figure 6.7 shows the sequence diagram in one clock cycle of using semaphores for the first synchronization. The semaphore and flag shown in the diagram belong only to this extra thread; that is, each extra thread gets its own semaphore and flag.

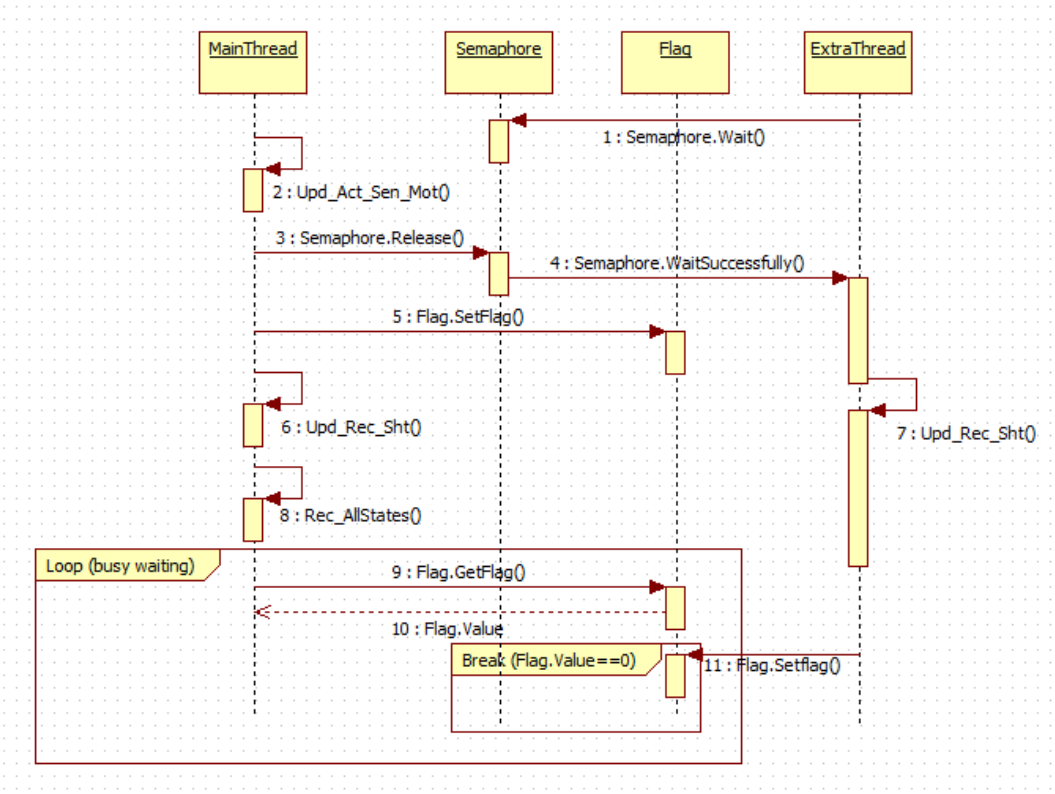

Figure 6.7: Sequence diagram of using semaphores.

2. Spin locks. Figure 6.8 shows the sequence diagram in one simulation clock cycle of using spin locks for the first synchronization.

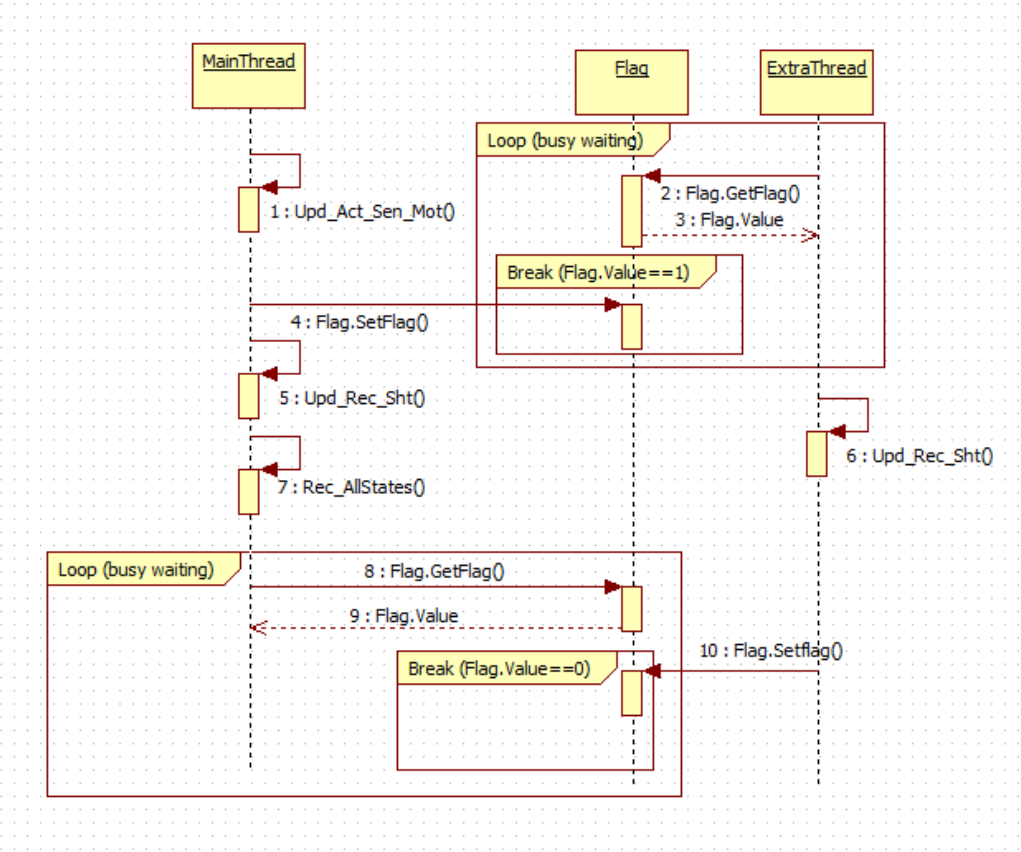

Figure 6.8: Sequence Diagram of using spin locks in one simulation clock cycle.

3. Mixed implementation, which uses both spin locks and semaphores. The aim of this implementation is trying to get a balanced design of CPU usage and performance. Figure 6.9 shows the sequence diagram of this design in one simulation clock cycle.

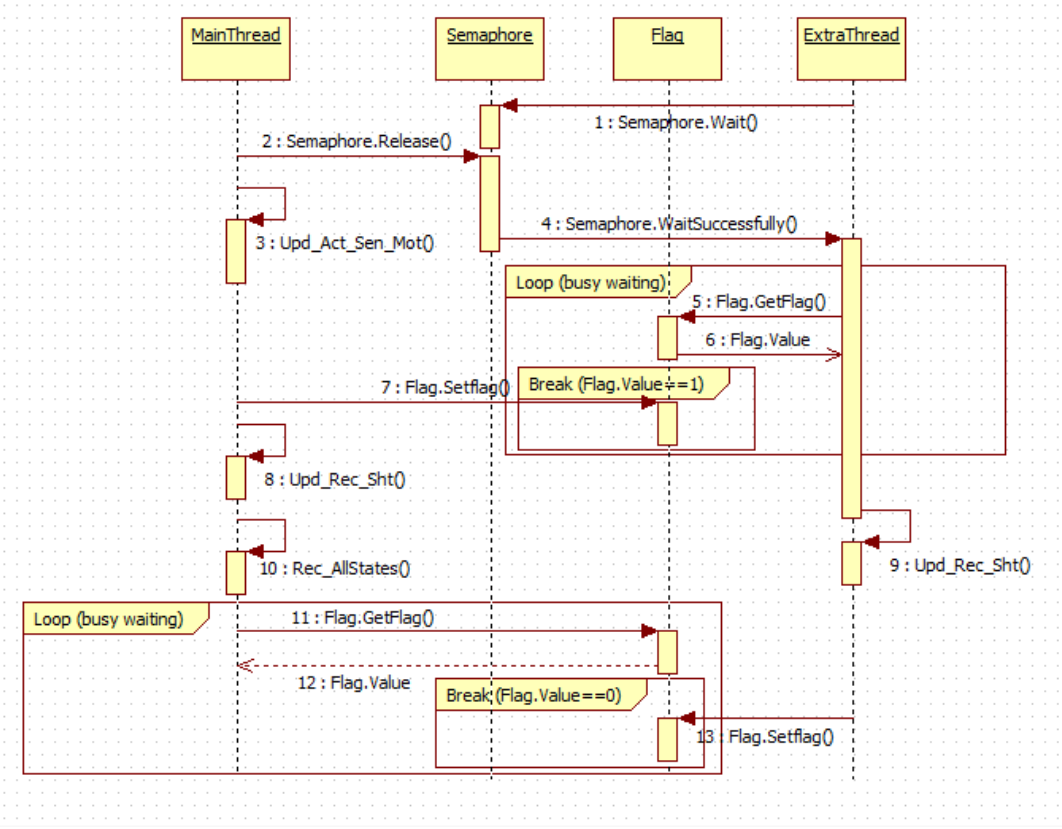

Figure 6.9: The sequence diagram of the mixed implementation in one simulation clock cycle.

Advantages: This implementation considers the trade-off between CPU usage and performance. It will hide part of the overhead of semaphores by updating other hardware, for example, actuators, sensors and motors in parallel. It will also block the extra threads when the main thread is out of the scope of SheetLogic. The average CPU usage is only a little higher than that of using semaphores, while the performance is a little worse than that of using spin locks.

Disadvantages: The performance is not stable. If there is some interference from outside, it will spend more time to wake up and run the extra threads in every simulation clock cycle. The best performance can only be archived without interference.

Decision:
For current SIL simulation, using spin locks can achieve the best performance. The mixed implementation has a better performance than using pure semaphores. The solution also depends on the external environment. If there is too much interference, mixed implementation should be used.

Rationale:
Upd_Rec_Sht is still a short period task. The overhead of using a semaphore will degrade the performance a lot. The overhead is even larger than the execution time of updating all the low level hardware. Although the mixed implementation tries to hide the overhead of using semaphores, it still

cannot hide all the overhead. On the other hand, using spin locks has almost no overhead. The cost is that it will have a very high CPU usage.

6.5 Number of threads

The number of physical threads is limited by the processor. To get the best performance, SIL simulation should take most advantage of the processor. If there are too many threads, a lot of overhead will be generated from the context switches between threads. An experiment has been done to explore the relationship between the number of extra threads for Sheet Manager and the total simulation time of printing 500 sheets in simplex mode. Figure 6.10 shows the total simulation time with using different extra threads.

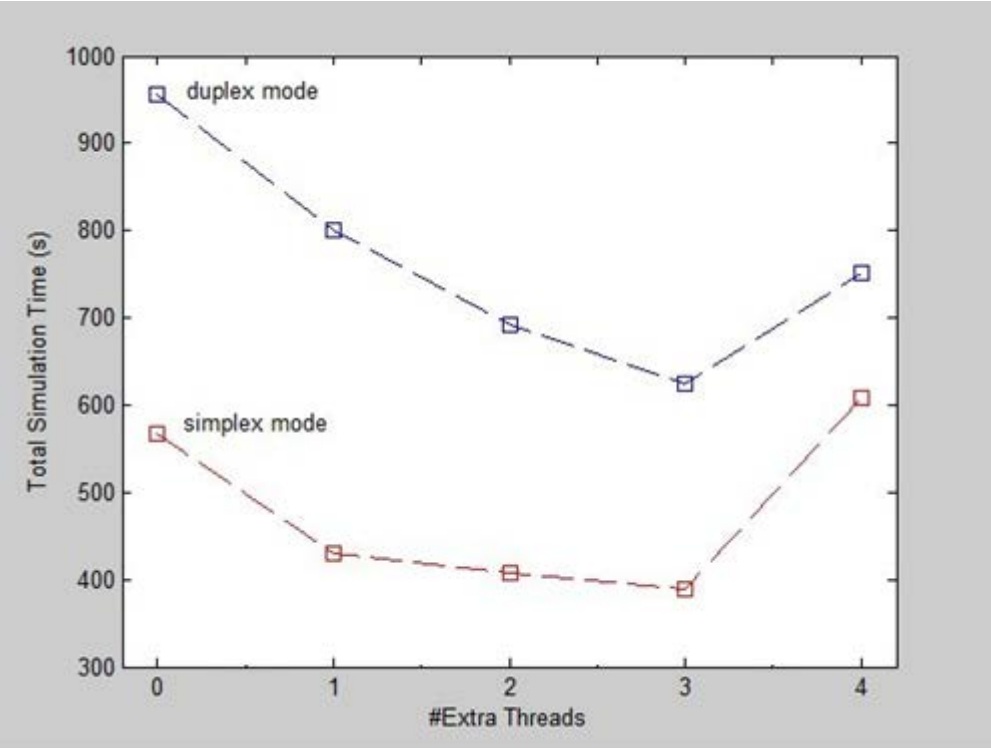

Figure 6.10: Total simulation time for printing 500 sheets with using different number of threads.

The experiment is done under Windows 7 Professional with Intel(R) Core(TM) i5-2400 CPU @ 3.10 GHz (4 cores and 4 threads), it is obvious that using three extra threads will get the best performance. These three extra threads and the main thread will consume about 95% of CPU. The reason of using 4 extra threads makes the performance worse is that too much context switch slows down the simulation.

6.6 Extent of parallelism

As discussed in Section 2.3, the Sheet Manager focuses on the sheets in the paper path. For a continuous printing job, the paper path is full of sheets most of the time. For example, Figure 6.11 shows the number of sheets in the paper path during the simulation in simplex mode of two printing jobs with 100 sheets and 10 sheets respectively.

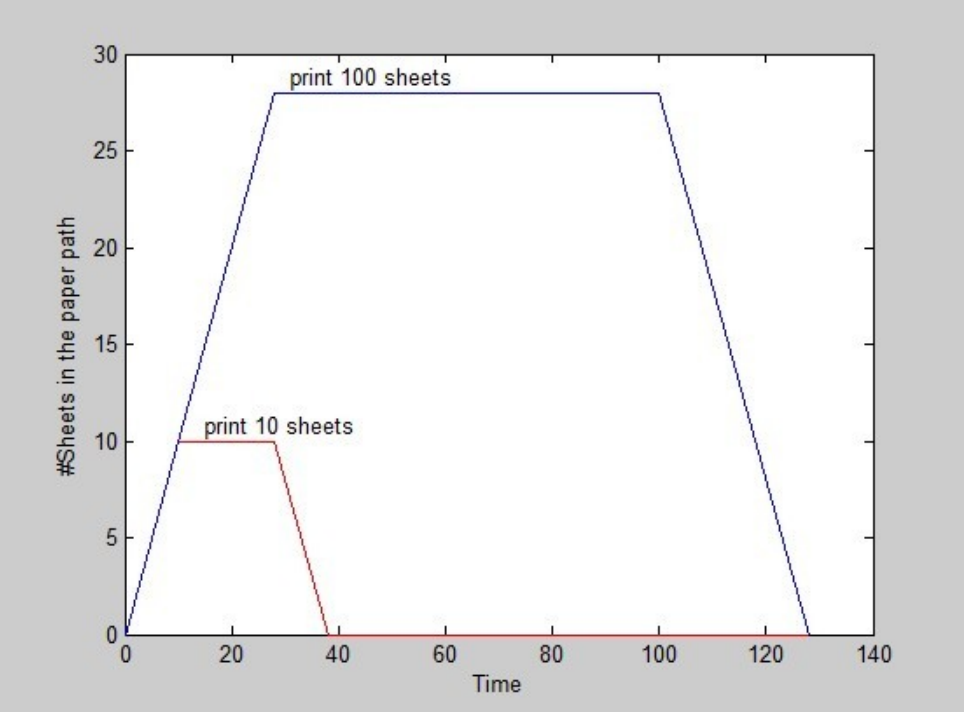

Figure 6.11: The number of sheets in the paper path for two printing jobs.

From the graph, it is obvious that there are not always a lot of sheets in the paper path, especially for small printing jobs. Actually, there are three possible cases when the number of sheets in the paper path is not large:

1. Small printing jobs. The number of sheets that need to be printed is not so large in contrast with the maximal number of sheets in the paper path.
2. Small printers. The maximal number of sheets in the paper path is small.
3. For big printers and also big printing jobs, the paper path is not full at the very beginning and at the end of the simulation.

For all the cases above, it is better to use the original sequential Sheet Manager because the overhead of multi-threading is larger than the benefit. For a simulation, the Sheet Manager will run as the original sequential one until the number of sheets in the paper path exceeds a certain number. In this way, small printers or small printing jobs would have the same performance as before. Otherwise, they will get a punishment from multi-threading. Figure 6.12 shows block diagram of the Sheet Manager.

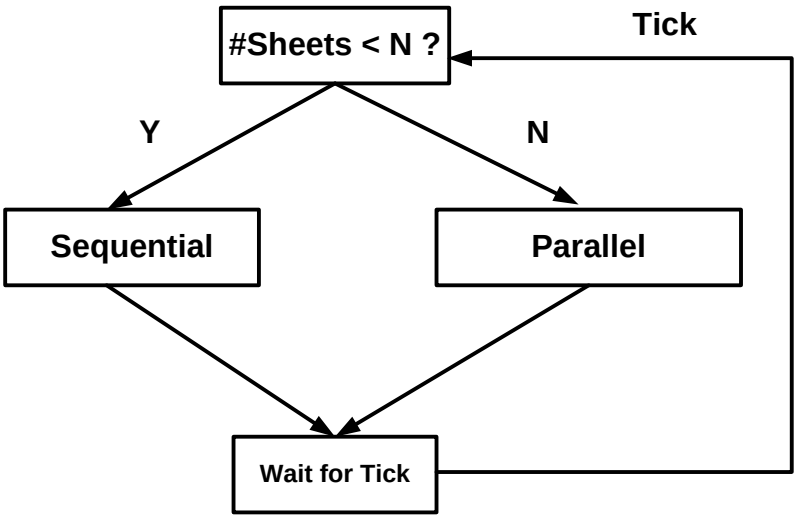

Figure 6.12: Mixed structure of Sheet Manager.

This approach helps the mixed implementation more. When the number of sheets in the paper path is low, it is necessary to get rid of the overhead of semaphores. Some experiments have been done and the results show the optimal value of N is 4 for X1. The optimal value of N is determined by many factors, for example, the hardware of the computer used for simulations and the type of sheets that should be simulated.

6.7 Race conditions of threads

When multiple threads want to access the flags of locks, the race conditions should be taken into account. To make the process safe enough, it is necessary to make sure that only one thread can access the variable at one time because every thread will change the value of the flag after fetching it. To avoid the race conditions, each extra thread has its own flag and semaphore which can only be used by this thread and the main thread. In each simulation clock cycle, the extra threads will only change the value of flags and semaphores which belongs to them. This assures the exclusive access to the flags and semaphores.

7 Result and data analysis

In this chapter, the performance of SIL simulation is investigated. Firstly, a proper definition for time is given. Then the performance improvement is presented in the following sections. The data analysis is done after the results have been presented.

7.1 Timing definition

As discussed in Section 2.6, the most important standard to evaluate the performance is the total simulation time.

Total simulation time is the total time spent for simulate a printing job. If the printer is already in “Standby” state, the timer will start when the printing job comes. When the virtual printer finishes flushing and goes back to “Standby” state, the timer will also stop at this moment. The total simulation time consists of two parts: preparation time and printing time.

The preparation time is determined by the model of the printer, the hardware which the simulation runs on, and simplex mode or duplex mode. It is fixed no matter how many sheets will be simulated. The output delay, which starts from the first sheet entering into the printer and ends when the first sheet gets out of the printer, can also be considered as a part of preparation. The output delay is fixed to the length of the paper path. Since the printing time for one sheet is very small, the total simulation time of a printing job of one sheet can be measured as the preparation time.

After the first sheet gets out of the paper path, all the other sheets in this printing job will be printed in a pipelining style. The printing time of the other sheets would be linear with the number of sheets that need to be printed. In this assignment, we can only decrease this printing time, which is a part of simulation time.

7.2 Experiment setup

As introduced in Section 2.5, the virtual printer will be waiting in the “Standby” state. When the printing job comes, the timer will start to measure the simulation time. When the virtual printer finishes flushing and goes back to “Standby” state, the timer will also stop at this moment. The timer will use the clock of the operating system and the precision is one second. The timer is located in the remote controller.

Now the SIL simulation runs on a multi-core processor, it may be interfered by other programs in Windows. The simulation time may vary. The best case comes with the shortest execution time while the worst case comes with the longest execution time. The best case can be only achieved with the least interference and it can be used to determine the maximal acceleration. Here the average is used to represent the expectation of the simulation from the end users. The average simulation time is the average of ten runs. At the same time, the deviation of ten times simulation will be calculated to evaluate the stability of the performance.

The experiment is done on the Intel i5-2400 @ 3.10GHz. This processor has four cores and each core contains one physical thread. The operating system is Windows 7 Professional.

7.3 One sheet

As discussed before, the total simulation time of printing one sheet is approximately the preparation time. The preparation time cannot decrease because all the implementation is done based on the sheets. During the preparation, there is no sheet. And the printing time of one sheet also cannot decrease because only one sheet cannot be distributed over different threads. Normally, the main thread will update and record more sheets than extra threads. The reason is that the main thread does not suffer any overhead because it stays active. If the main thread is used to update this sheet, the total simulation time should be the same as before.

But as discussed in Section 6.3, if this sheet is updated and recorded by the extra thread, it can run in parallel with the main thread which is recording the states of all the hardware. So the performance can be improved. Figure 7.1 shows the total simulation time of one sheet in both simplex and duplex mode for spin locks implementation and also mixed implementation.

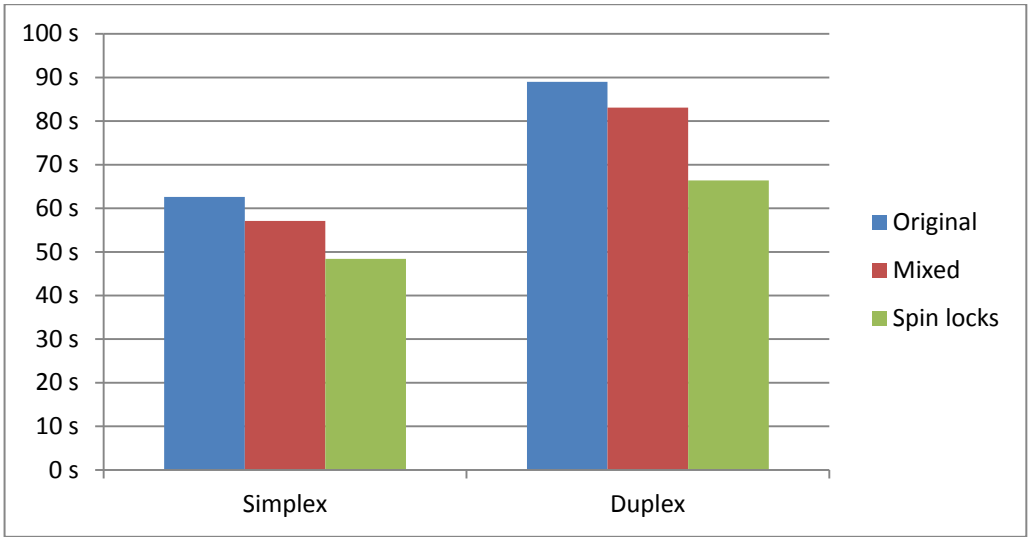

Figure 7.1: The total simulation time of one sheet in both simplex and duplex mode for spin locks implementation and also mixed implementation.

These results can only be obtained if the extent of parallelism is always set to parallel in Figure 6.10. Otherwise, the Sheet Manager will become to sequential when there is only one sheet in the paper path. Then it will have the same performance as before. At this moment, it is complicated to merge this solution to the implementations. But in the future, it will take the place of sequential part in Figure 6.10. So the Sheet Manager will switch the way of parallelism based on the number of sheets in the paper path.

7.4 500 sheets and 2000 sheets

A printing job with 500 sheets is a typical test case for the SIL simulation. Figure 7.2 shows the total simulation time of 500 sheets in both simplex mode and duplex mode for spin locks implementation and also mixed implementation.

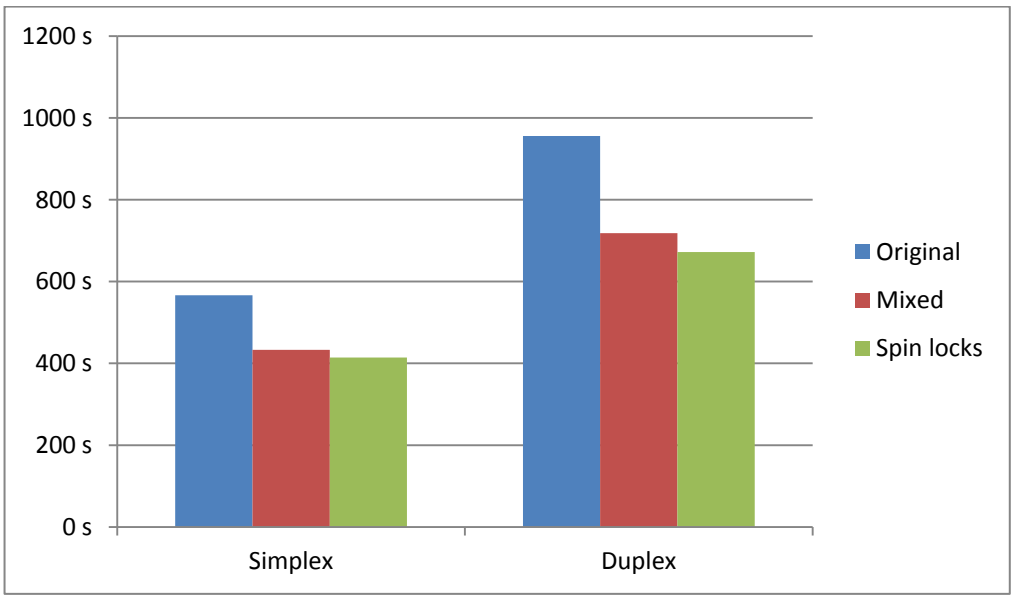

Figure 7.2: The total simulation time of 500 sheets in both simplex mode and duplex mode for spin locks implementation and also mixed implementation.

It is a big job to print 2000 sheets. Since the preparation time is fixed, the more sheets are printed, the proportion of the printing time will be bigger, the more precise the performance gain can be seen. Figure 7.3 shows the total simulation time of 2000 sheets in both simplex mode and duplex mode for spin locks implementation and also mixed implementation.

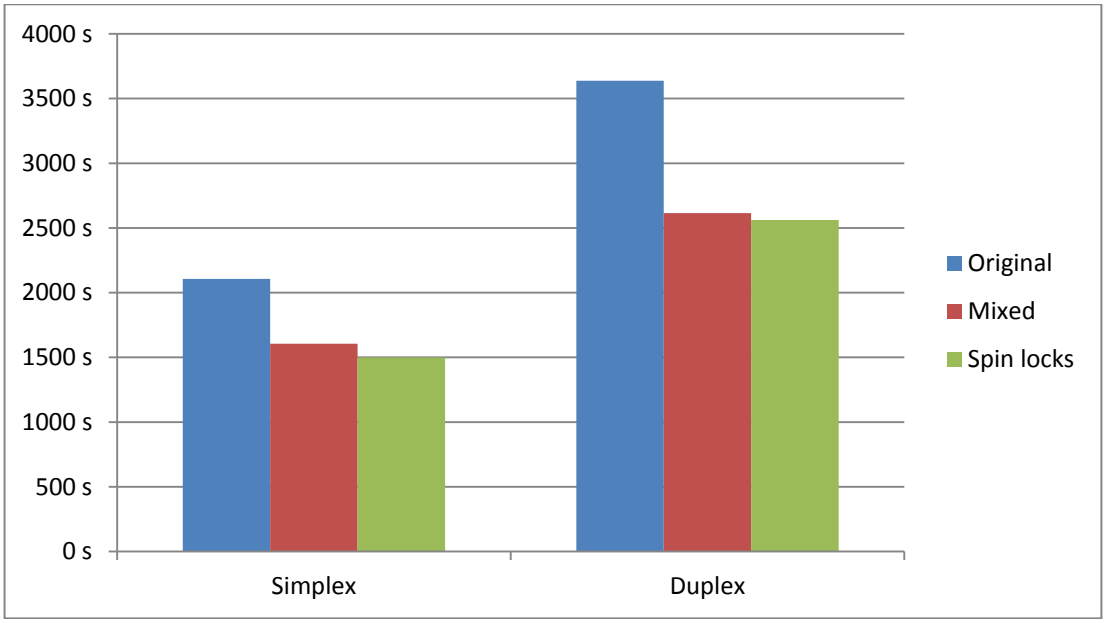

Figure 7.3: The total simulation time of 2000 sheets in both simplex mode and duplex mode for spin locks implementation and also mixed implementation.

7.5 Acceleration for two implementations

As discussed in Section 6.4, the performance of mixed implementation is worse than that of spin locks because of the overhead of using semaphores. Figure 7.4 shows the comparison of acceleration between mixed implementation and spin locks implementation with the reference of original throughput.

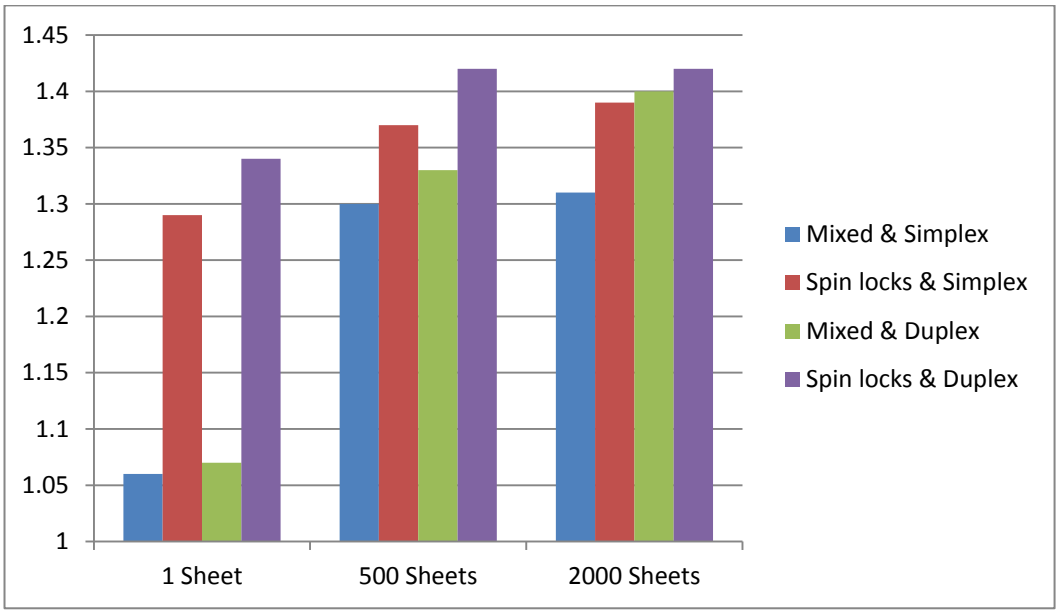

Figure 7.4: Higher acceleration of performance archived by spin locks implementation.

For small jobs, the difference of performance between two implementations is more obvious. The reason is that the main thread of SheetLogic has no information when will the job come, so it will release the semaphores in every simulation clock cycle after initialization. During the preparation time, releasing semaphores and the overhead of using semaphores will slow down the performance a little for several seconds. It is obvious in small jobs when the total execution time of small jobs is less than 100 seconds.

For big jobs, the difference of performance becomes smaller. The difference here is mainly caused by the overhead of using semaphores in every simulation cycles in the printing time.

7.6 Acceleration for jobs with different sizes

As discussed in Section 7.1, a part of the preparation time cannot be reduced. A job which has more sheets to be printed will have more benefit since the printing time will have a greater proportion. Figure 7.5 shows the acceleration of mixed implementation for jobs with different sizes in both simplex mode and duplex mode while Figure 7.6 shows the acceleration of spin locks implementation of jobs with different sizes.

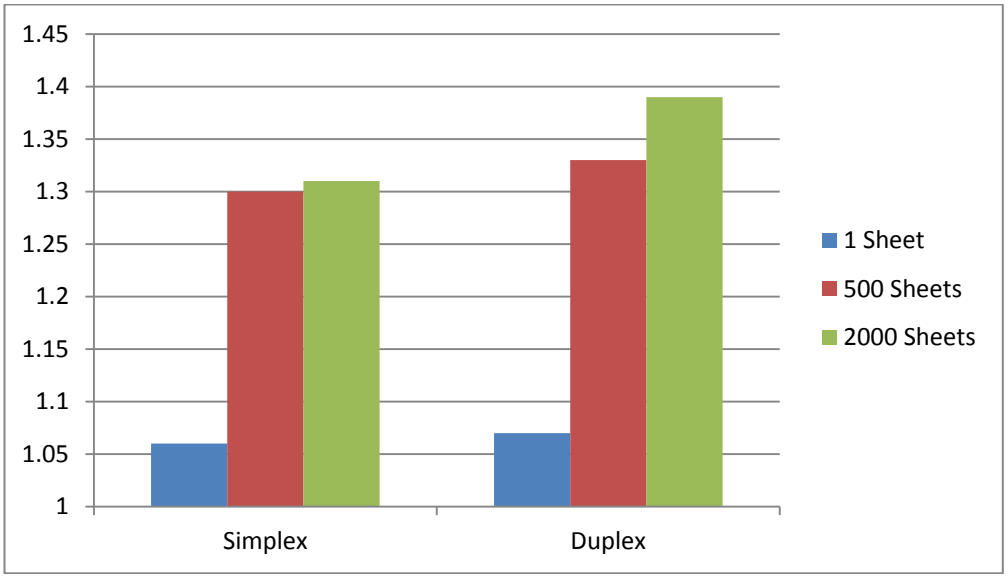

Figure 7.5: Acceleration of mixed implementation for jobs with different sizes in both simplex mode and duplex mode.

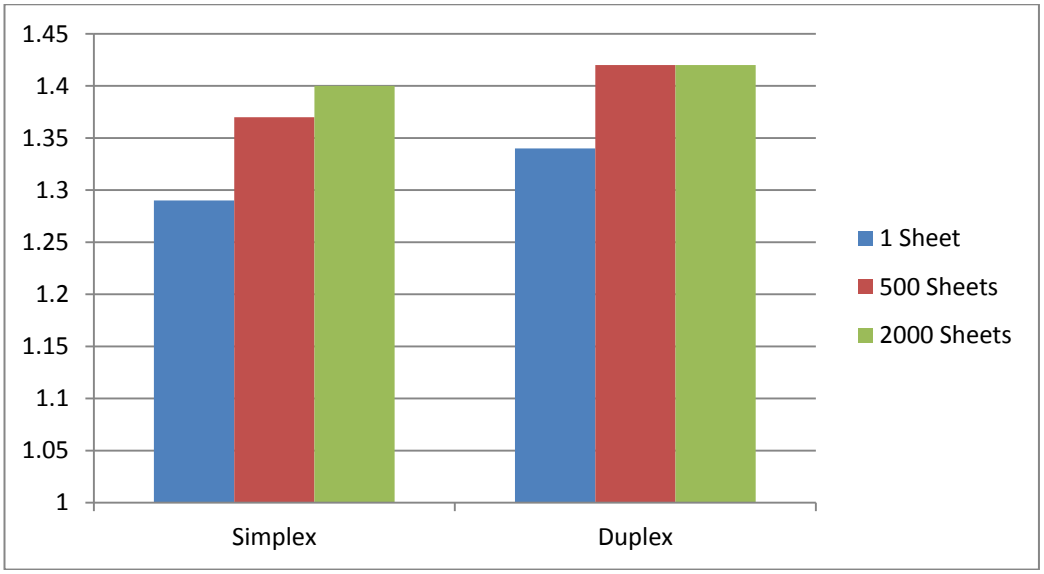

Figure 7.6: Acceleration of spin locks implementation for jobs with different sizes in both simplex mode and duplex mode.

From the results, it can be concluded that the performance can be improved more if there are more sheets to be printed. But when the number of sheets is increasing, the performance improvement will increase slower and slower. Then some more experiments have been done. It shows that if the number of sheets is beyond 2000, the acceleration will stay the same as before. For large jobs, the printing time is already much larger than the preparation time so the printing time can be considered as the total simulation time. Since the acceleration for the printing time is more or less fixed, the acceleration for the total simulation time would also be fixed.

7.7 Acceleration for simplex mode and duplex mode

The difference between simplex mode and duplex mode is the length of the paper path. In duplex mode, the paper path is longer than that in simplex mode. A longer paper path can contain more sheets. In every simulation clock cycle, the Sheet Manager will update all the sheets in the paper path. So every extra thread will update more sheets in one simulation clock cycle. Since the overhead will stay the same in the simplex mode and duplex mode, an increased computation load can improve the efficiency which means the performance of SIL simulation can improve more in duplex mode. Figure 7.7 shows the acceleration difference between simplex mode and duplex mode.

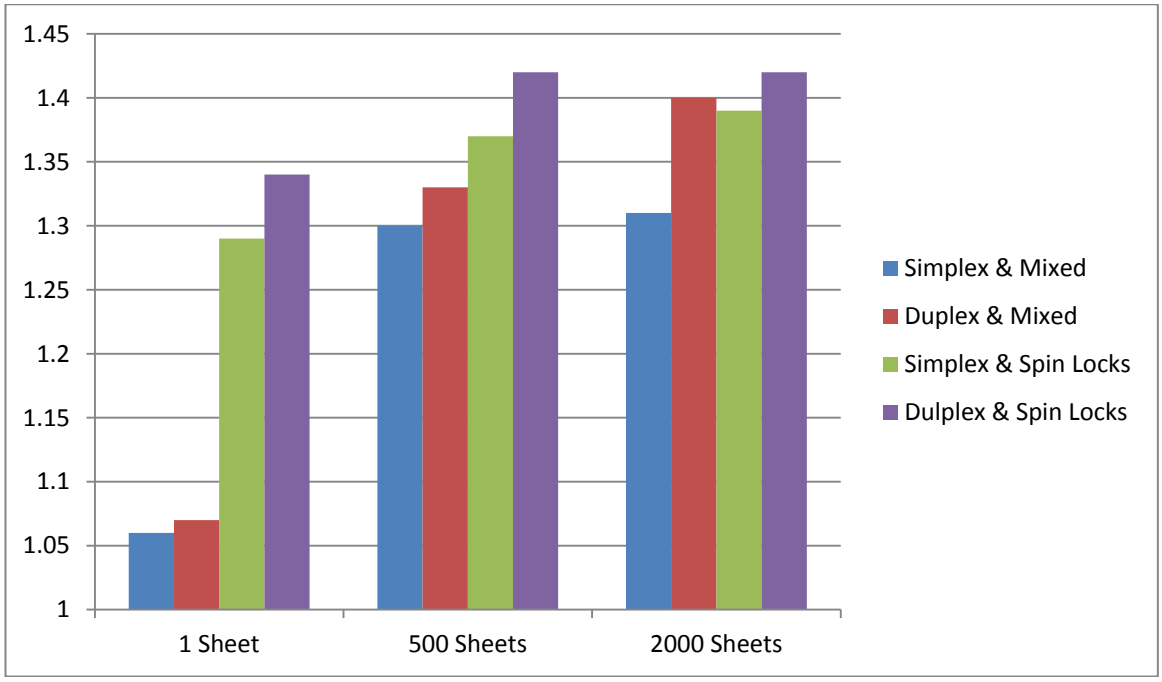

Figure 7.7: Higher acceleration in duplex mode in both implementations.

It is obvious that there is more performance improvement in duplex mode than that in simplex mode no matter with which implementation. It proves that the longer the paper path is, the more performance improvement can be gained. In other words, the multi-threading implementation is more suitable for big printers with a long paper path.

7.8 Deviations within ten times simulation

As discussed before, the total simulation time of the same jobs may vary due to the interference between the threads. For the spin locks implementation, all the extra threads created by the Sheet Manager will stay active and continue using CPU. After they become active in the initialization, it receives only a little interference from other threads for context switching. But for the mixed implementation, the extra threads will have an uncertain waiting time to let all the ready threads finish before them. Although some latency is hidden, the performance is still highly influenced by other threads.

Here the results of ten simulations have been collected. The total simulation time is $T_1, T_2, \ldots, T_{10}$. The average of them is used to represent the expectation of total simulation time.

$$E = \frac{\sum_{i=1}^{n} T_i}{n}$$

And the standard deviation is used to represent the stability of performance.

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(T_i - E)^2}$$

Here the results of ten simulations are collected, so *n*=10.

Figure 7.8 shows the standard deviation among ten simulations for both implementations for 500 sheets and 2000 sheets.

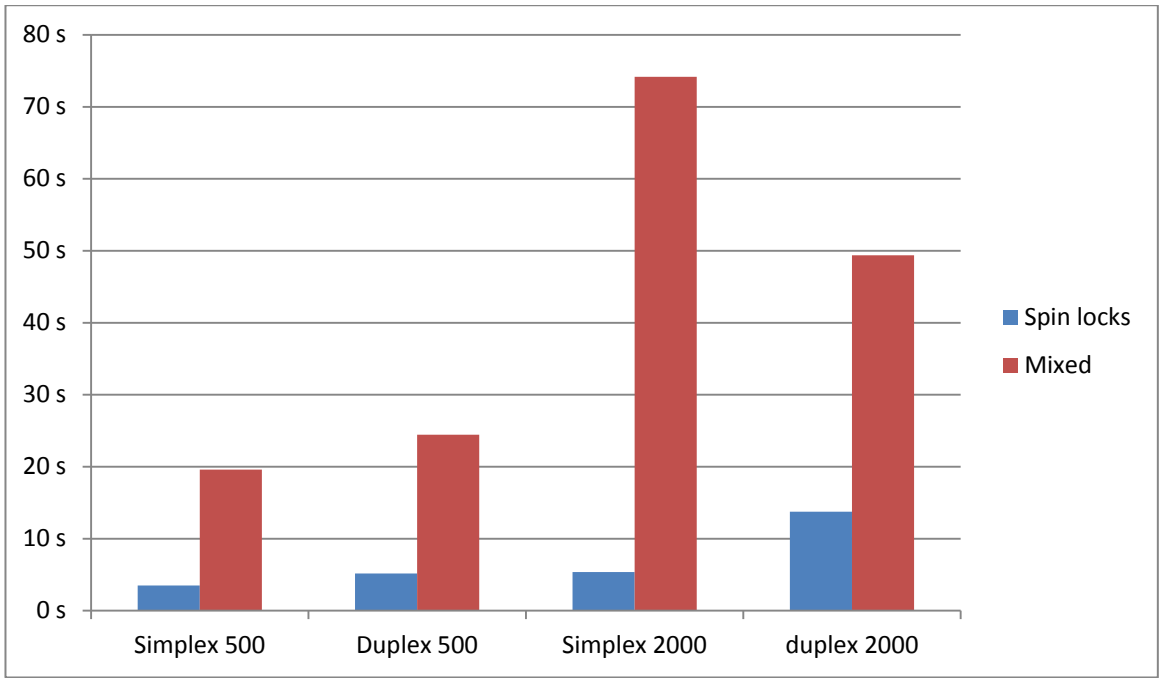

Figure 7.8: The variations among ten simulations of different jobs for two implementations.

As expected before, the standard deviation in mixed implementation is much larger than that in the spin locks implementation.

7.9 CPU usage

There are about 200 threads running in parallel in SIL simulation. But the load is not balanced. The main thread has the most of the computation load. Other threads will use idle cores for a very short period. It can be considered as a single-thread application. The average CPU usage of the original implementation is about 28%, which is a little higher than a quarter.

For the implementation with spin locks, the main threads and three extra run on different cores. They will take advantage of all four cores except for the CPU time used by the operating system. The average CPU usage is about 97% to 98%, which is almost all the CPU time.

The mixed implementation will block the extra threads when the main thread gets out of the scope of SheetLogic. It can decrease the average CPU usage compared with spin locks. The average CPU usage is about 45%.

7.10 500 sheets for X2

X2 is the next generation of the Model X1 printer. At the same time, the SIL simulation environment is being further developed, becoming more complex. The functionalities of sub nodes have been extended. The threads created by every sub node increased from 2 to more than 20. Normally, there are more than 20 sub nodes in the printer and all the sub nodes are triggered by the simulation clock. The number of threads has increased by several hundreds. It is difficult for the main node to know whether all the sub nodes have already been “scheduled”. To overcome this problem, SIL will create an “idle thread” for each core which has the lowest priority. After all the threads finish their tasks, the “idle thread” can get access to the CPU. Then the main node will know all the sub nodes have been “scheduled”. This process is called “Idle Synchronization”.

It has two potential problems with the multi-threading implementation:

(1) There are much more threads running in parallel in SIL simulation now. The interference between threads will be more. The overhead of context switches may slow down the performance again.
(2) The multi-threading sheet manager may have some conflicts with “Idle synchronization”. The “idle thread” may suffer a longer waiting time because the priority of threads created by Sheet Manager is higher than that of “idle thread”.

To test whether both the mixed and spin locks implementations work for the new model and simulation environment, and have similar impact on performance, 500 sheets are simulated to be printed in both simplex mode and duplex mode for the original implementation, mixed implementation and spin locks implementation.

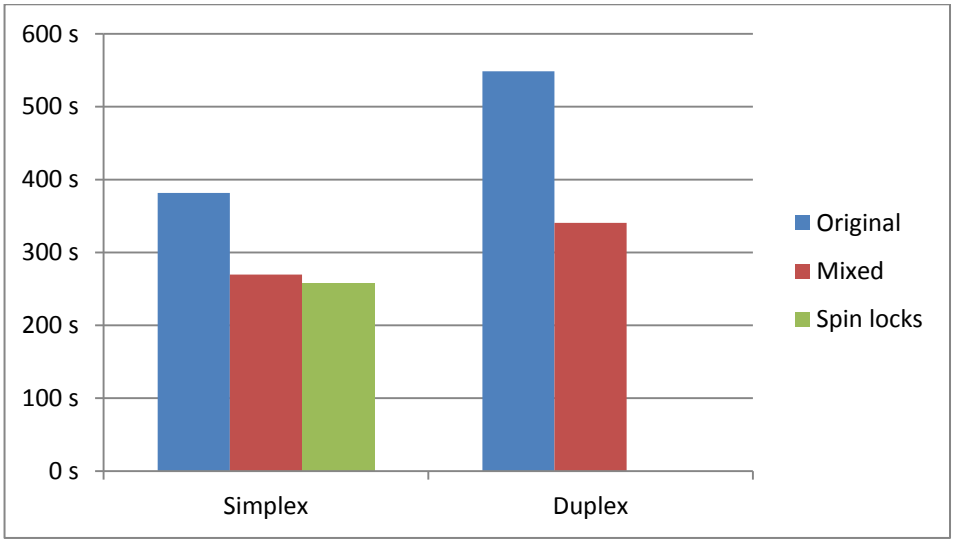

Figure 7.9: The total simulation time of 500 sheets for X2 simulation in both simplex mode and duplex mode for the original implementation, mixed implementation and spin locks implementation.

In X2 simulations, there are some errors with the simulation of low level hardware. This may be the reason that using spin locks implementation cannot print 500 sheets successfully in duplex mode. The original implementation and mixed implementation also suffer from this error; for example sometimes the simulation has to stop with this error. The results used here are obtained from the successful simulations.

From this graph, it is obvious that both multi-threading implementations improve the performance a lot. The acceleration rate from the original throughput for the simulation of 500 sheets is shown in Figure 7.10.

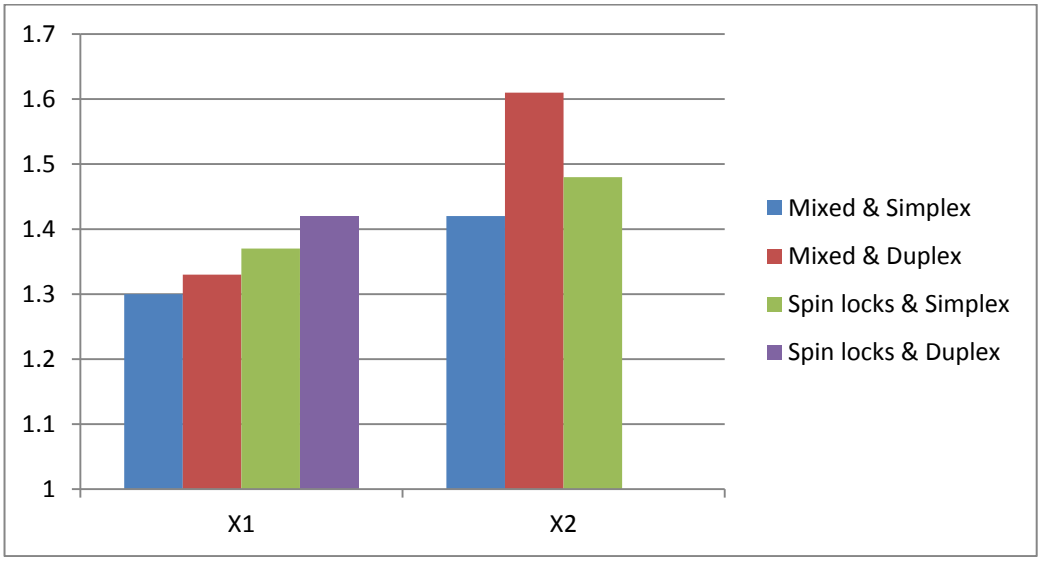

Figure 7.10: The acceleration rate of both X1 and X2 for the simulation of 500 sheets.

From the figure, it is apparent that the acceleration rate in X2 is even higher than that in X1. It can be concluded that the multi-threading implementation has no conflict with the next generation of SIL simulation.

Figure 7.11 shows the standard deviation of mixed implementation for X1 and X2.

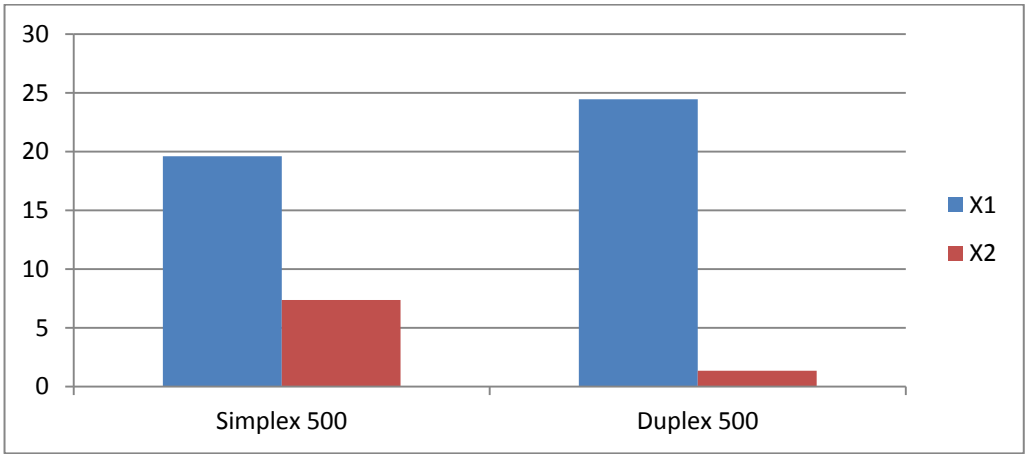

Figure 7.11: Standard deviation of mixed implementation for X1 and X2.

The deviation in X1 is much larger than that in the X2 model. The deviation in X2 is acceptable. Because the simulation environment, for example the hardware of the computer and the operating system, is the same, but the deviation in X2 decreased, it can be concluded that the interference must come from other threads in SIL rather than outside. The reason why the interference decreased is the changes in the SIL simulation.

7.11 Performance with visualization

As introduced in Section 2.1, the visualization front-end of SIL is called Argus. Argus provides different functionalities to visualize the state of SheetLogic. The properties of the different parts (sensors, actuators, pinches, etc) can be visualized in a list or in a 3D environment. Argus also provides functionality to manipulate the simulation by, for example, changing the values of sensors or actuators or stopping sheets. This is done via the so-called command interface. Visualization seriously degrades the performance of SIL simulation. The information needed by Argus comes from SheetLogic. SheetLogic has to update all the information in every simulation clock cycle. Table 7.1 shows the performance with/without visualization for printing 500 sheets in simplex mode.

Visualization	Total Simulation Time (s)
Disabled	566.6
Enabled	975.9

Table 7.1: The total simulation time of printing 500 sheets in simplex mode with/without visualization.

Apparently, the performance will degrade a lot if visualization is enabled. The multi-threading Sheet Manager will improve the performance, but there are two interferences which we should take into consideration.

(1) The visualization capsule in SIL. In every simulation clock cycle, the visualization capsule is also triggered by the simulation clock. There are two threads created by the visualization capsule. If the visualization is disabled, these two threads will be blocked. The visualization capsule partly depends on SheetLogic because it needs the current information of SheetLogic. It is not easy to figure out how much interference is generated by the visualization capsule.

(2) Argus. As the visualization tool, it should run in parallel with the SIL simulation. If the operating system is not overloaded, Argus will use approximately 15% of the CPU time.

To get the best performance, the number of threads created by the Sheet Manager should decrease to get rid of the overhead of context switch between threads. Some experiments are done to explore the best solution. Figure 7.12 shows the performance with using different numbers of threads when visualization is enabled for both mixed implementation and spin locks implementation.

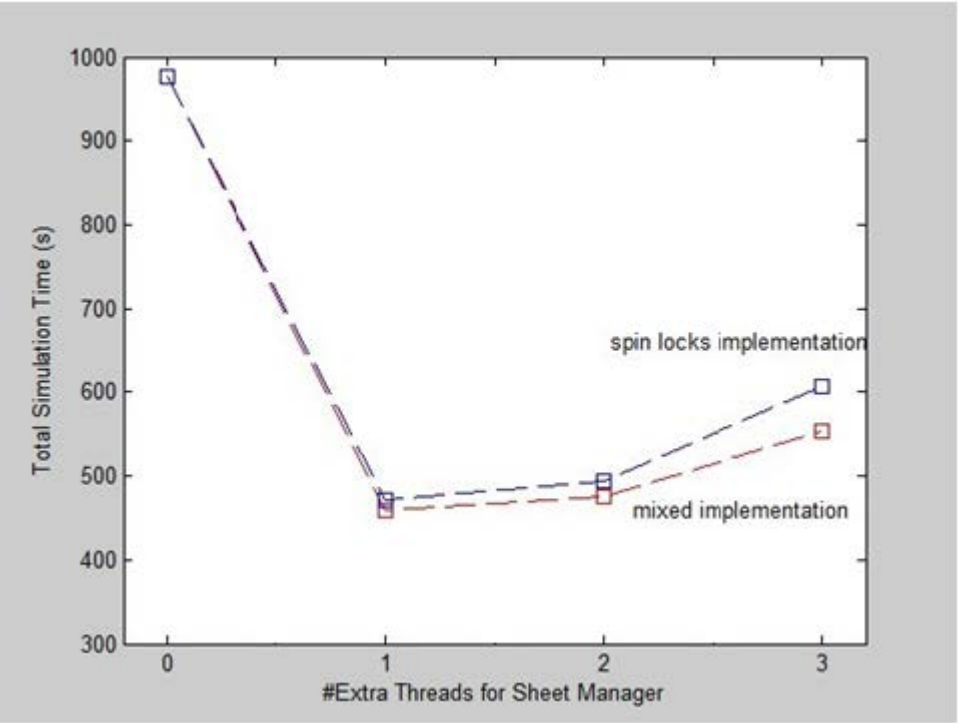

Figure 7.12: The total simulation time for printing 500 sheets in simplex mode with using different number of threads when visualization is enabled.

From the results, it is obvious that using three extra threads is not optimal for the performance any more. Using one or two extra threads for the Sheet Manager will get the best performance. It is important to have a close look at these two approaches.

Using one extra thread for the Sheet Manager will bring less benefit to the SheetLogic. The main thread and the extra threads will have more computation load to update and record sheets compared with using two extra threads. But this approach only uses one extra core, so it will eliminate a lot of context switching. Since there is little interference between threads, the performance is also more stable than that by using two extra threads. Table 7.2 lists the standard deviation of using one and two extra threads when visualization is enabled.

Standard Deviation (s)	Mixed Implementation	Spin Locks Implementation
One Extra thread	1.34	1.51
Two Extra threads	24.23	8.73

Table 7.2: The standard deviation of using one and two extra threads for both implementations.

The standard deviation of using one extra thread is much less and acceptable. Besides, the performance is also better. It can be concluded that using one extra thread for the Sheet Manager is the best choice when visualization is enabled.

It is also easy to explain why the performance of mixed implementation is better than that of spin locks implementation. In spin locks implementation, the extra threads created by the Sheet Manager will stay active, which will lead to more context switches.

8 Case studies

This chapter presents some case studies which are relevant with this project but also can be discussed as independent issues. The first section presents the method to check the correctness of Sheet Manager. Then the overhead of semaphores and spin locks are profiled by the performance profiler to be compared. In the end of this chapter, a method which can decrease the load of scheduler is discussed.

8.1 MatLab script to check the correctness of Sheet Manager

The task of the graduation project is to improve the performance of Sheet Manager. During the process, the code must be modified. It is important to preserve the correctness of Sheet Manager.

It is hard to say whether the code before and after changes is consistent by looking at the results of printing jobs shown in Remote Controller. A piece of incorrect code may cause some small deviations to the paper path but all the sheets can still be printed successfully. The solution given in this report solves this problem by analyzing the behaviour of each sheet.

8.1.1 Behaviour of Sheets

The behaviour of a sheet is recorded in the log file of this sheet. The records are event-triggered rather than time-triggered. When some attributes of the sheet change, the log file will be written to record what happened to this sheet and also the current state of this sheet. Figure 8.1 shows a part of the log file of a sheet.

Sheet(0)

absTime	relTime	dT[ms]	event	LE_X[mm]	dX[mm]	[mm/s]	LE_Pinch	TE_X[mm]	dX[mm]	[mm/s]	TE_Pinch	len[mm]	nrP	nrS
71179.00	0.00	0.00	>>P_INPMINPPI1	0.00	0.00	0.00	-	-210.00	0.00	0.00	-	210.00	1	0
71180.00	1.00	1.00	*spd_change*	2.40	2.40	2399.98	INPMINPPI1	-207.60	2.40	2399.98	INPMINPPI1	210.00	1	0
71182.00	3.00	2.00	*spd_change*	4.80	2.40	1199.99	INPMINPPI1	-205.20	2.40	1199.99	INPMINPPI1	210.00	1	0
71184.00	5.00	2.00	>>SE_Engine/INPMINPSE	4.80	0.00	1199.99	INPMINPPI1	-205.20	0.00	1199.99	INPMINPPI1	210.00	1	1
71316.00	137.00	132.00	>>P_INPMINPPI2	163.20	158.40	1199.99	INPMINPPI1	-46.80	158.40	1199.99	INPMINPPI1	210.00	2	1
71352.00	173.00	36.00	<P_INPMINPPI1	206.40	43.20	1199.99	INPMINPPI2	-3.60	43.20	1199.99	INPMINPPI1	210.00	1	1
71360.00	181.00	8.00	<SE_Engine/INPMINPSE	218.40	12.00	1199.99	INPMINPPI2	8.40	12.00	1199.99	INPMINPPI2	210.00	1	1
71386.00	207.00	26.00	>>P_INPMINPPI3	247.20	28.80	1199.99	INPMINPPI2	37.20	28.80	1199.99	INPMINPPI2	210.00	2	0
71408.00	229.00	22.00	>>SE_Engine/INPMINPXCRSE	273.60	26.40	1199.99	INPMINPPI3	63.60	26.40	1199.99	INPMINPPI2	210.00	2	1
71490.00	311.00	82.00	<P_INPMINPPI2	372.00	98.40	1199.99	INPMINPPI3	162.00	98.40	1199.99	INPMINPPI2	210.00	1	1
71526.00	347.00	36.00	>>P_INPMINPREGPI3	415.20	43.20	1199.99	INPMINPPI3	205.20	43.20	1199.99	INPMINPPI3	210.00	2	1
71528.00	349.00	2.00	*spd_change*	420.00	4.80	1200.03	INPMINPREGPI3	210.00	4.80	1199.99	INPMINPPI3	210.00	2	1
71560.00	381.00	32.00	<P_INPMINPPI3	456.00	36.00	1200.03	INPMINPREGPI3	246.00	36.00	1200.60	INPMINPPI3	210.00	1	1
71560.00	381.00	0.00	*spd_change*	458.40	2.40	1200.03	INPMINPREGPI3	248.40	2.40	1200.60	INPMINPPI3	210.00	1	1
71562.00	383.00	2.00	*spd_change*	460.80	2.40	1200.03	INPMINPREGPI3	250.80	2.40	1200.03	INPMINPREGPI3	210.00	1	1
71566.00	387.00	4.00	*spd_change*	465.66	4.86	1230.03	INPMINPREGPI3	255.66	4.86	1230.03	INPMINPREGPI3	210.00	1	1
71568.00	389.00	2.00	*spd_change*	468.16	2.50	1249.99	INPMINPREGPI3	258.16	2.50	1249.99	INPMINPREGPI3	210.00	1	1
71582.00	403.00	14.00	<SE_Engine/INPMINPXCRSE	485.66	17.50	1249.99	INPMINPREGPI3	275.66	17.50	1249.99	INPMINPREGPI3	210.00	1	1
71662.00	483.00	80.00	>>P_INPMINPREGPI2	583.16	97.50	1249.99	INPMINPREGPI3	373.16	97.50	1249.99	INPMINPREGPI3	210.00	2	0
71696.00	517.00	34.00	<P_INPMINPREGPI3	625.66	42.50	1249.99	INPMINPREGPI2	415.66	42.50	1249.99	INPMINPREGPI3	210.00	1	0
71798.00	619.00	102.00	>>P_INPMINPREGPI1	753.16	127.50	1249.99	INPMINPREGPI2	543.16	127.50	1249.99	INPMINPREGPI2	210.00	2	0

Figure 8.1: An example of a sheet's log file

Table 8.1 explains the contents of the log file.

Abbreviations	Meaning
absTime	Absolute simulation time.
relTime	Relative simulation time. The timer started when the sheet entered the virtual printer.
dT	The time difference from the last record.
event	The name of the event happened to this sheet.

LE_X	The displacement of the leading edge from the beginning of the paper path.
dx[1]	The displacement of the leading edge from the last record.
mm/s[1]	The average speed of the leading edge between two events.
LE_Pinch	The name of the pinch which handles the leading edge of this sheet.
TE_X	The displacement of the terminal edge from the beginning of the paper path.
dx[2]	The displacement of the terminal edge from the last record.
mm/s[2]	The average speed of the terminal edge between two events.
TE_Pinch	The name of the pinch which handles the terminal edge of this sheet.
Len	The absolute length of the sheet. If the sheet is curly, the length will decrease.
nrP	The number of pinches connected to this sheet.
nrS	The number of sensors connected to this sheet.

Table 8.1: The contents of a sheet's log file

One sheet will be printed in about 6000 simulation clock cycles in simplex mode. In this period, the number of events happened to this sheet is around 460. The events that happen to a sheet should be the same every time a job is simulated, but in practice they are influenced by the Embedded Software. This will lead to the same sheets in different simulations having a different displacement at the same relative time point. It becomes difficult to say whether the results of a simulation after changes have been made to the code are correct. So, the regular pattern of the behaviour of sheets should be found out. Then the regular pattern can be used to check whether the results of a new simulation can be regarded as correct.

8.1.2 Regular Pattern

The regular pattern is built on the relationship between the displacement of the LE or TE with relative time. Figure 8.2 shows the relationship between LE displacement and relative time. The lines show the first sheets' behaviour (LE displacement) in two simulations for 100 sheets.

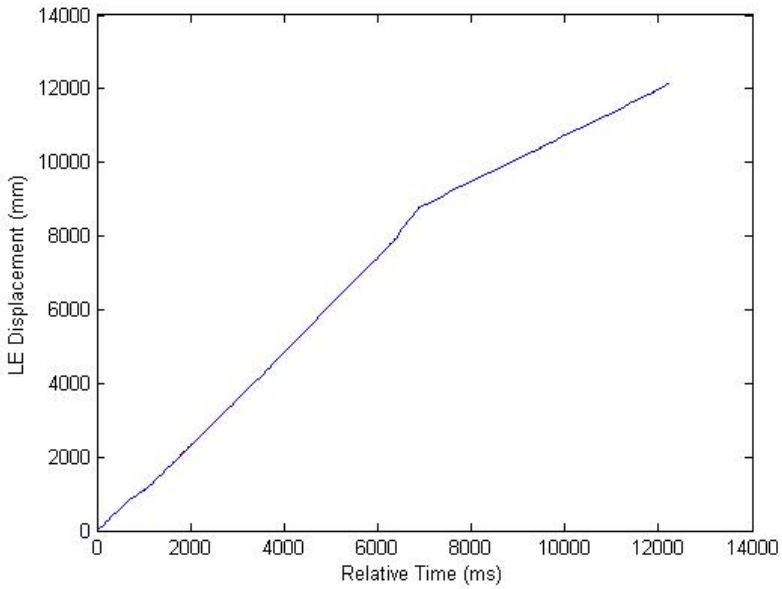

Figure 8.2: First sheets' behaviour in two simulations for 100 sheets.

In Figure 8.2, there is only little difference between the behaviours of these two sheets. The reason is that the sheet should arrive at the next segments of paper path at the right time. If the arrival time is not as desired, the Embedded Software will adjust the sheet. The deviation will be eliminated sooner or later. If we zoom into this graph, some small differences can be found between these two sheets. Figure 8.3 shows the enlarged graph.

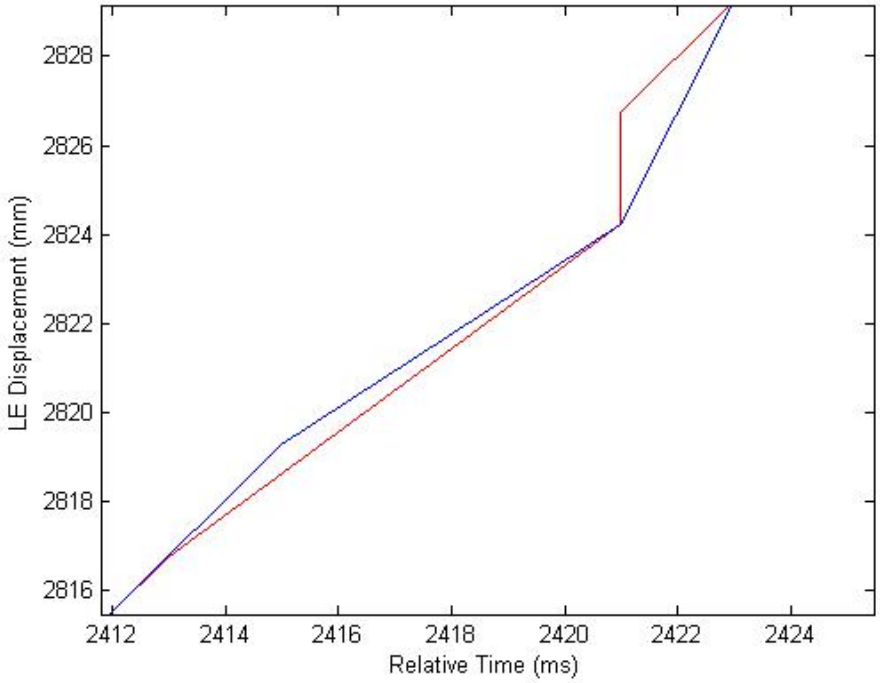

Figure 8.3: Small differences between two simulations of the same job when looking at the LE displacement of the first page.

The difference between two sheets is defined by using quadratic mean. Assume there are two simulations Sim_1, Sim_2 with m sheets. The sheets in Sim_1 are represented as $s_1, s_2, s_3, \dots s_m$ while the sheets in Sim_2 are represented as $s_1', s_2', s_3', \dots s_m'$. And the x^{th} event of sheet i can be denoted as e_x^i. Here LE/TE displacements are taken from the events. Then the difference of the same sheets in two different simulations can be calculated. n is the number of events happened for the sheets.

$$D_{s_i,s_{i'}} = \frac{1}{\sqrt{n}} \sqrt{\sum_{k=1}^{n} (e_k^i - e_k^{i'})^2}$$

However, this formula can only be applied to ideal situations where the simulation clock frequency is high enough. If the simulation clock frequency is not high enough, some events may happen at the same time. An example is given in Figure 8.4.

```
72242.00    1063.00     14.00  <P_INPMINPREGSPI          1137.87    10.51   875.71
72242.00    1063.00      0.00 *spd_change*             1139.62     1.75   875.71
```

Figure 8.4: Events happening at the same time.

When the relative time is 1063ms, two events happened and are recorded as two separated records in the log file. The displacements of the sheets when these two events happened are different, which means a sheet has two different positions at the same time. This is impossible in a real printer but it exists in a virtual printer because of the low simulation clock frequency. And there are also some exceptions that some events only happened to some sheets. So before comparing two sheets, it is necessary to process the events of sheets with the following strategy.

1. If more than one event happened at the same time, these events will be merged into one event and the new LE/TE displacements are the average of those old events.
2. If some events happened to one sheet rather than the other one in a comparison, these events would be removed in this comparison. In some extreme cases, about 10% of the events only happened to one sheet. For most cases, none or just 1% of the events happened to only one sheet. In this way, the difference from Simulation 2 to Simulation 1 will be strictly equal to the difference from Simulation 1 to Simulation 2.

After processing the event lists, comparison can be done with slightly changes with the previous formula.

$$D_{s_i,s_{i'}} = \frac{1}{\sqrt{l}} \sqrt{\sum_{k=1}^{l} (e_k^i - e_k^{i'})^2}$$

where l is the length of the event list after processing. ($l \leq n$)

After calculating all the differences of the same sheets in two simulations, an array of differences can be obtained.

$$D_{Sim_1,Sim_2} = [D_{s_1,s_{1'}},\ D_{s_2,s_{2'}},\ D_{s_3,s_{3'}},\ \dots\ D_{s_m,s_{m'}}]$$

Some attributes of the array can be easily obtained. For example, the difference between two sheets, the mean difference between two simulations and also the trend of the curve are important attributes of the comparisons.

8.1.3 Experiments

To find out which attributes can be used to check the results, some experiments are taken. Here 50 simulations of the same job are done and the results have been collected. Then the results of all simulations have been compared to each other. So the number of comparison is $C_{50}^2 = 1225$.

Table 8.2 shows the results of comparisons.

Comparisons	Results
LE maximal difference between two sheets	1.8082
TE maximal difference between two sheets	1.8967
Maximal mean LE difference between two simulations	0.2793
Maximal mean TE difference between two simulations	0.2808

Table 8.2: Result of comparisons.

The results in Table 3.1 shows the worst cases of difference between two sheets in different simulations and also the mean difference of two simulations. The best case for two sheets in different simulations is 0. The best case for two simulations should be 0 but no absolutely same simulation has been found until now. Some simulations have a small difference which is less than 0.0001. So we can accept the best case for two simulations can be thought as 0.

Figure 8.5 shows the curve of difference between 2 pairs of simulations.

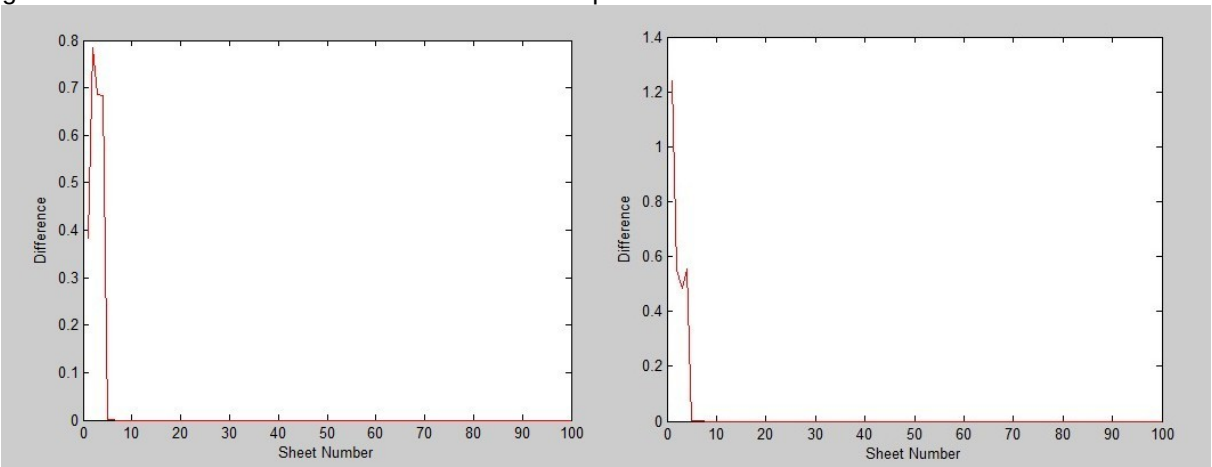

Figure 8.5: The difference curve between simulations (LE displacements).

The differences for the first several sheets are higher than the following ones. The shapes of curves are quite similar to each other.

8.1.4 Correctness Checking

Based on the analysis of the comparisons, the results of a new simulation can be checked in this way to determine whether the Sheet Manager is still correct after modification:

(1) Compare the results with the existing results of 50 simulations.
(2) LE maximal difference should be less than or equal to 0.2793.
(3) TE maximal difference should be less than or equal to 0.2808.
(4) The maximal mean LE difference from any other simulations should be less than or equal to 1.8082.
(5) The maximal mean TE difference from any other simulations should be less than or equal to 1.8967.

This can be done by scripts automatically. If all the requirements are satisfied, the Sheet Manager can be thought of as correct. If some requirements are unsatisfied, a further step would also help to get some hints:

(6) Draw the difference curve compared with other simulations. If there is big difference after the 10^{th} sheet, it is probably wrong.

It is also possible that the results of a new simulation cannot satisfy the requirements but it is still correct. Since the number of existing simulations is limited, we cannot guarantee some extreme cases can be treated correctly.

The execution time of the script is about 5 to 6 minutes. Since the correctness checking should be done step by step to find out the errors as soon as possible, the simulation log file should be collected after any changes to the code. If there are several changes in one day, it is better to check the correctness at night with the log file from every step.

8.2 Overhead of spin locks and semaphores

In this section, some experiments have been done to measure the overhead of spin locks and semaphores. In a multi-threading program like SIL simulation, it is hard to measure the execution time for the line level or function level. Using an independent timer thread does not work any more because some other threads with higher priorities may pre-empt the timer thread. Then the execution time which is obtained by the timer thread is often less than the real execution time. To get the precise execution time, a professional performance profiler is needed.

8.2.1 AQTime

AQTime is a professional profiler produced by SmartBear. It can be used to profile both 32-bit and 64-bit applications. It monitors the application execution and gathers considerable information about each application function, for instance, the number of function calls, the hierarchy of function calls, time spent executing the function, etc. Figure 8.6 is the user interface of AQTime 8. Here AQTime is used to

measure the elapsed time spent on each thread T_{thread} without the execution time of children as overhead. The execution time of children is the calculation time.

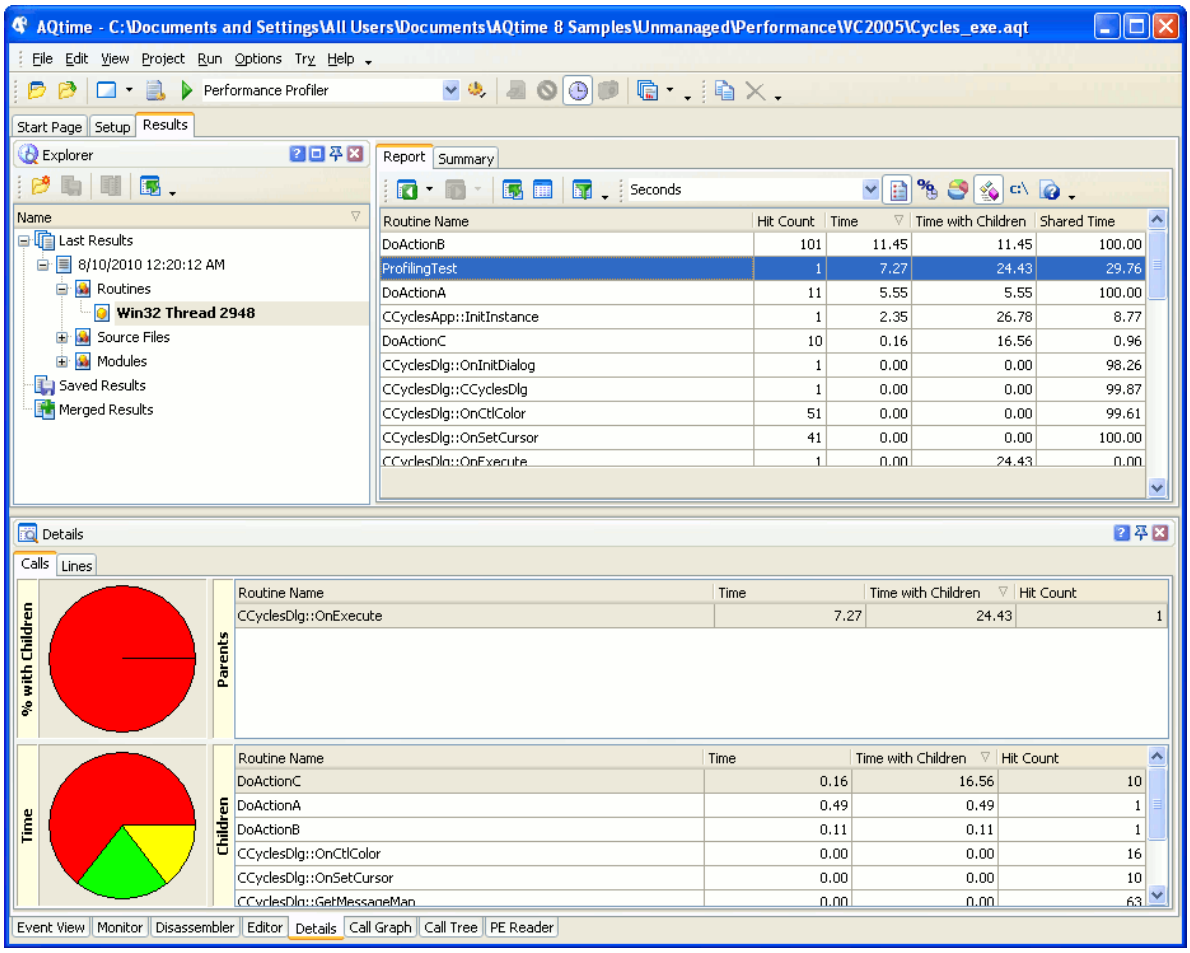

Figure 8.6: The user interface of AQTime 8.

8.2.2 Experiment setup

The aim of the experiment is to measure the overhead of using different approaches for synchronization. To give a similar environment like SIL simulation, a new model with one capsule is built in Rose RealTime. The new model only has one capsule, TopLevelCapsule. In this capsule, there are two states and one transition between them. Figure 8.7 shows the state diagram of this capsule.

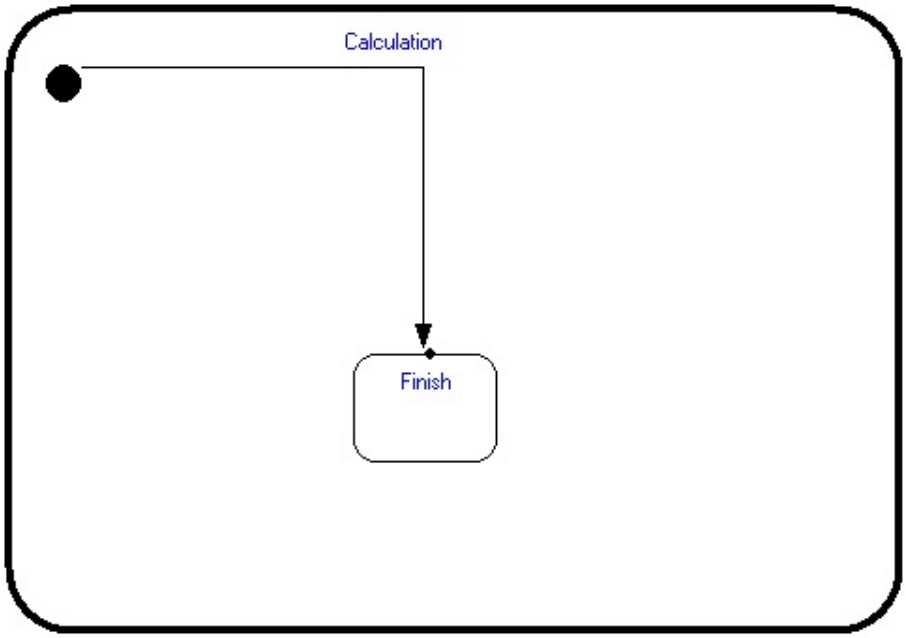

Figure 8.7: The state diagram of the capsule in the model for experiment.

In the transition “Calculation”, a loop for 100000 times is the body of the transition. The loop can be considered as the simulation clock in the SIL simulation. In the SIL simulation, the clock signal is also event-triggered. There are two synchronizations in every SIL simulation clock cycle while there are also two synchronizations in the loop for this model. In this model, the second synchronization, spin locks on the main thread, is the same as that in SIL simulation. For the first synchronization, here semaphores and spin locks are used and then the overhead of them is measured by AQTime.

8.2.3 Results and analysis

Three extra threads and one main thread are used here to simulate the Sheet Manager in SIL simulation. T_{thread} may be different for each thread. To get the bottleneck, the longest one among three is used to determine the overhead of synchronization. Because this model is also multi-threaded, then the average of five runs is used to express the expectation. Table 8.3 shows the elapsed time of threads without their children.

	Thread	Elapsed Time (us)	Elapsed Time per Cycle (us)
Semaphores	Extra Thread	946862	9.46862
	Main Thread	245921	2.45921
Spin Locks	Extra Thread	293154	2.93154

Table 8.3: Elapsed time of threads without their children for semaphores and spin locks.

For semaphores, the overhead on the extra thread is the waiting time that comes from the operating system's scheduler. The overhead on the main thread is the execution time to release the semaphores. The spin locks implementation does not need to release semaphores, so it has little overhead on the main thread. And it has less overhead on the extra threads since the extra threads stay active while running.

The experiment can be used to show that the overhead of semaphores is more than that of spin locks. But the elapsed time cannot be used to estimate the performance in other models. The overhead that comes from the scheduler is not fixed. It is decided by the system, the hardware and also the current load of the operating system scheduler.

8.3 Setting the affinities of threads

The reason why the semaphore is more expensive than spin locks is that the overhead from the scheduler increase significantly when there are a number of threads in parallel. Every thread will contribute some load to the scheduler. One way to decrease the load is setting the affinity of threads. After setting the affinity of one thread with a core, the thread will only run on that core. Then the scheduler will not consider this thread. It can decrease the waiting time for other threads.

For both of the implementations, the Sheet Manager will create three extra threads. If the affinities of these threads are set to different cores, it will bring some benefit:

(1) It can decrease some load to the operating system scheduler. Since there are hundreds of threads in SIL simulation, the user may not see any significant improvement of performance.
(2) By setting different extra threads to different cores, the extra threads are exclusive form each other.
(3) The positions of extra threads are known, it can decrease the randomness which will make the performance more stable.

Table 8.4 lists the performance and standard deviations of two different mixed implementations, one with setting affinities while the other without. The experiment is done with 500 sheets in simplex mode.

Setting Affinities	Total Simulation Time (s)	Standard Deviation (s)
No	432.9	19.59
Yes	433.6	3.36

Table 8.4: The performance and standard deviations before/after setting the affinities.

The performance is more or less the same as before. If the affinities have been set, it can improve some performance by decreasing the load of scheduler. But these threads will also get penalty because of the affinities. If one extra thread is waiting for its core and other cores are already standby, it will keep waiting for its core. Setting affinities will decrease the standard deviation dramatically which means the performance becomes more stable.

Setting affinities will also bring several disadvantages:

(1) As the extra thread will only run on its core, if this core is always occupied by other program or the operating system, the performance would be very bad.
(2) The scalability is bad. Before running the SIL simulation, the information about processors is needed.

Due to the reasons above, I suggest that it is better not to set affinities for extra threads.

9 Future Work

In this project, the performance of SIL simulation has been improved. However, there is still some future work to be done to improve performance more or get a better understanding of the performance.

(1) Now the implementation is done based on the processor with four cores. If this SIL runs on a processor with two cores, it will be much slower than using only one extra thread. It is necessary to do the automation of threads creation based on the number of available processors. If visualization is disabled, the Sheet Manager will create $N-1$ extra threads to update and record sheets, where N is the number of cores in the processor on which the SIL simulation is running.
(2) If the automation of threads creation is done, there is still a problem. As discussed in Section 7.11, if visualization is enabled, creating $N-1$ threads is not optimal anymore. The solution is that Sheet Manager or SheetLogic should keep an eye on the visualization capsule in SIL. If the visualization capsule is enabled, it is better to create $N-3$ or $N-2$ extra threads for the Sheet Manager. This approach needs the modification of the structure of the SIL core because the dependency on the visualization capsule to SheetLogic capsule should be added.
(3) The module adapter capsule should be improved. The module adapter is A_Node in Figure 2.2. This capsule is also triggered by the simulation clock. In the configuration of X1, there are more than 20 sub nodes. All the sub nodes are distributed on 16 threads. Actually, the execution time of each thread is very small. It is not wise to use so many threads because the overhead of context switches may be even larger than the execution time. It will slow down the execution of SheetLogic because so many threads need the CPU at the same time. It is worth to figure out what is the best way to incarnate these sub nodes to threads.
(4) Until now there is no standard test case for performance. To get a better understanding of the performance, it is necessary to design a test case for performance. Then detailed profiling should be done by using AQTime to know the performance of each part. After any implementation of SIL simulation has changed, the performance should be a non-functional requirement to be discussed.
(5) In MSDN library, the condition variable can be also used to lock the thread. It is necessary to compare the performance between condition variable and semaphore and then the one with a better performance should be used. Since Ration Rose Real Time with Mingw cannot compile the kernel library, the experiment should be done in another environment.

As the performance of SIL simulation will influence the efficiency of the development process of Embedded Software, it should be taken into consideration anytime by the software developer.

10 Conclusion

The aim of the graduation project was to improve the performance of the SIL simulation by SheetLogic. This has been done via two important ways, namely a smarter algorithm and a multi-threading implementation. The performance has been improved significantly especially when the visualization is enabled.

The smarter algorithm can be applied to any printer model because it is independent of the length of the paper path. It separates the sheets by their positions. Then the Sheet Manager can update and record the sheets in the paper path. The algorithm will bring more benefits when there are more sheets to be simulated.

The multi-threading implementation is very sensitive to the overhead because the execution time of the tasks in the tick transition is short. It is not possible to create threads in every simulation clock cycle so that the extra threads should be reused from the beginning. Two possible implementations which can improve the performance are a spin locks implementation and a mixed implementation using spin locks and semaphores. For both of the implementations, the latency generated by the synchronization should be hidden as much as possible. Spin locks implementation has a better and more stable performance, but it will also have an extremely high CPU usage which is near 100%. The mixed implementation makes a trade-off between the CPU usage and the performance. The performance is a little worse than the spin locks implementation but it still has a big improvement. The CPU usage decreases from 100% to about 45% for a four-core processor. Due the interference, the performance is not as stable as spin locks implementation. Both of the implementations can improve the throughput by at least 30% for 500 sheets in simplex mode and by at least 33% for 500 sheets in duplex mode. And both of the implementations can improve the performance more when there are more sheets to be simulated.

The visualization part is also a very important factor for the performance. For the original version of the SIL simulation, the visualization degrades the performance of the SIL simulation a lot. Running the SIL simulation with visualization means the total simulation time will double. And when the visualization is enabled, the visualization capsule in SIL and Argus will generate two heavy threads. So for a four-core processor, the best solution for the SIL simulation with visualization is that only two cores are used for the Sheet Manager while the other two cores are reserved by the visualization. In this way, the throughput of simulating 500 sheets in simplex mode with visualization can be improved more than 100% for both of the implementations.

During the development process, a MatLab script has been worked out to check the correctness of the Sheet Manager. This script can also be used to check the behaviours of the Sheet Manager in other printer models.

The method of software development process is agile. Iterative and incremental development is used along the whole project to decrease the risk. In each step, the Sheet Manager was checked to make sure the correctness.

11 Reference

[1] Rational Software Cooperation, “C++ Reference Rational Rose @ Real Time”. ftp://ftp.software.ibm.com/software/rational/docs/v2003/win_solutions/rational_rosert/rosert_cpp_ref_guide.pdf.
[2] Christian Terwellen. “Evaluating the behavior of embedded control software, using a module software-in-the-loop simulation environment and a domain specific modeling language for plant modeling”. Unpublished master's thesis, Océ Technologies, 2012.
[3] S. van der Hoest. “The development of a software-in-the-loop simulation framework for testing real-time control software”. SAI technical report, Eindhoven University of Technology, August 2006.
[4] Laurens Vrijnsen. “Architecture of the Virtual Printer”. Project report, Océ Technologies, January 2011.
[5] Rational Software Cooperation, “C++ Porting Guide Rational Rose @ Real time”. ftp://ftp.software.ibm.com/software/rational/docs/v2002_r2/unix/pdf/RoseRT/c_portingguide.pdf.
[6] Henri Hunnekens. “SIL plan”. Project manual, Océ Technologies, 2012.
[7] Marius Muresan and Dan Pitica. “Software in the Loop Environment Reliability for Testing Embedded Code”, in *Proc*. Design and Technology in Electronic Packaging (SIITME), 25-28 Oct. 2012, pp. 325-328.
[8] Claudio Bonivento, Matteo Cacciari, Andrea Paoli and Matteo Sartini. “Rapid prototyping of automated manufacturing systems by software-in-the-loop simulation”, in *Proc*. Control and Decision Conference (CCDC), 2011 China, pp. 3968-3973.
[9] Scott Meyers. Effective C++ (Third Edition). ADDISON-WESLEY, 2005.
[10] David E. Culler, Michial Gunter, James C. Lee. “Analysis of Multithreaded Microprocessors under Multiprogramming” in Multithreaded computer architecture: summary of the state of the art. Kluwer, 1992, pp. 351-367.
[11] Gary Tan and Viv Woods. “Load Balancing and Multiprogramming in The Flagship Parallel Reduction Machine”, in *Proc*. Databases, Parallel Architectures and their Applications, 1990, pp. 560.
[12] Evangelos Markatos, Mark Crovella, Prakash Das. “The Effects of Multiprogramming on Barrier Synchronization”, *in Proc*. Parallel and Distributed Processing, 1991, pp. 662-669.
[13] Robert Love. Linux Kernel Development (Second Edition). Sams Publishing, 2005.

12 Appendix

12.1 Model X1 specification

Task ID	Function Name	Hardware Involved
1	Sleep(v_delay)	-
2	v_SheetLogicModuleAdapter.updateValues()	-
3	v_SheetLogicIOLayer.updateActuators()	67 simple actuators and 26 analog actuators
4	Actuator::updateActuators()	90 actuators.
5	Sensor::updateSensors()	217 sensors.
6	Motor::updateMotors()	70 motors, motor sensors and connected pinches.
7	D_SheetManager:: Instance()->updateSheets()	-
8	D_Tray::updateTrays()	-
9	f_UpdatePois()	-
10	D_Device::updateDevices()	-
11	Sensor::recordAllStates()	217 sensors.
12	Actuator::recordAllStates()	90 actuators.
13	Motor::recordAllStates()	70 motors.
14	D_Pinch::recordAllstates()	107 pinches.
15	D_SheetManager:: Instance()->recordAllStates()	-
16	v_SheetLogicIOLayer.updateSensors()	67 simple sensors and 46 analog sensors.
17	v_SheetLogicModuleAdapter.writeValues()	-
18	p_Clock.done().send()	-

Table 8.1: Model X1 specification with SheetLogic.

12.2 Timing service in Rational Rose Real Time

The execution time (elapsed time) can be obtained in the following way:

```
RTTimespec t1,t2,t;
static int count = 0;
double sum = 0.0, time = 0.0;
RTTimespec::getclock(t1);
/************************
Code to be measured
************************/
RTTimespec::getclock(t2);
t = t2 - t1;
time = t.tv_sec + t.tv_nsec/1000000000.0;
sum = sum + time;
count++;
if(count == NUM){
```

```
count = 0;
sum = 0.0;
printf(“%f”,sum/(double)NUM - deviation);
}
```

The execution time of the piece of code will be measured for *NUM* times. Then the average time will be calculated. It is important that the overhead of the timing services (deviation) should be eliminated to make the result more precise. The overhead depends on the computer and operating system. It can be obtained in this way:

```
RTTimespec t1,t2,t;
double deviation = 0.0;
RTTimespec::getclock(t1);
RTTimespec::getclock(t2);
t = t2 - t1;
deviation = t.tv_sec + t.tv_nsec/1000000000.0;
```

The service does not guarantee absolute accuracy. This means that intervals can take slightly longer than specified, and events scheduled for a particular time may in fact happen slightly after the actual time has occurred. The magnitude of the delay depends on many factors. However, unless the system is under severe overload, the discrepancy is usually not significant. (From [1])

Actually, if the multi-thread mode of Ration Rose is enabled, the SIL will assign an independent thread for the timing services. The timing services can be launched with little delay. Since there is almost no jitter in the timing services, the execution time recorded can be considered as real elapsed time. However, if the multi-threading Sheet Manager has been implemented, three extra threads will run in parallel with the main thread when the sheets are updated and recorded. This will make system under severe overload for a short time in every simulation clock cycle which will degrade the precision of measurement. So, the timing service can only be used before multi-threading has been implemented.