Experimental Analysis of Real-time Energy Control Potentials for Sustainable Manufacturing Systems

BY

HAOXIANG YANG
B.Eng., Dalian Jiaotong University, Dalian, P.R.China, 2010

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Industrial Engineering
in the Graduate College of the
University of Illinois at Chicago, 2012

Chicago, Illinois

Defense Committee:

Dr. Lin Li, Chair and Advisor
Dr. Houshang Darabi
Dr. David He

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Lin Li, Assistant Professor in the Department of Mechanical and Industrial Engineering, University of Illinois at Chicago, for his valuable guidance and support throughout the thesis project. No words can express my gratitude for his open of the door of Sustainable Manufacturing Systems Research Laboratory to me. I also wish to thank him for mentoring me not only academically but all other aspects which will be beneficial for my lifetime.

I would also like to thank Dr. Houshang Darabi and Dr. David He for serving as part of my graduate committee. Their valuable guidance during the project and other generous help are greatly appreciated.

I wish to thank all the other lab mates, Dr. Yong Wang, Mr. Zeyi Sun, Mr. Josh Reese, Mr. Chongye Wang, Ms. Mayela Fernandez, Mr. Erik Osland and Ms. Rubina Mirza. The study and life in the Sustainable Manufacturing Systems Research Laboratory could not be delighted without the collaboration with them.

Finally, I wish to thank my parents, Mr. Xingde Yang and Ms. Lifang Zhou. I would never have the opportunity to study in the United States without the unwavering support and assistance of them.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

FIGURE PAGE

LIST OF ABBREVIATIONS

API	Application Program Interface
CAN	Controller Area Network
DC	Direct Current
FLTK	The Fast, Light Toolkit
GHG	Greenhouse Gas
GNU	GNU's Not Unix
GUI	Graphical User Interface
IEA	International Energy Agency
JAGUAR	Stellaris® Brushed DC Motor Control Module with CAN (MDL-BDC24)
MCU	Microcontroller Unit
MES	Manufacturing Execution System
MTBF	Mean Time Between Failures
MTTR	Mean Time To Repair
OOP	Object-Oriented Programming
VBA	Visual Basic for Applications

SUMMARY

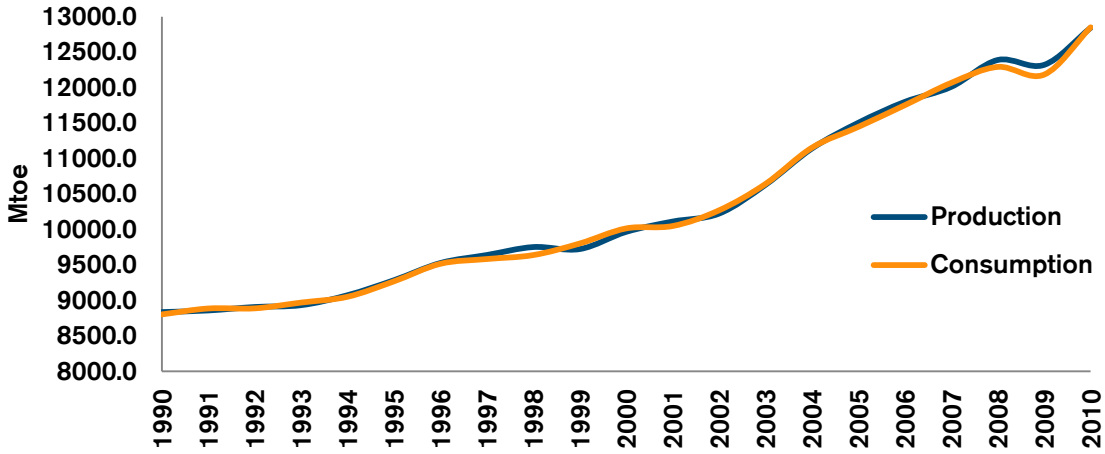
Energy efficiency improvement as well as carbon footprint reduction in modern manufacturing systems has been of high interests to both academia and industry in recent years. However, most existing research efforts in energy efficiency improvement only focus on either single machine or process level due to the complexity of modern manufacturing systems, and few literatures concentrating on the energy efficiency improvement for the typical manufacturing systems with multiple machines and buffers can be found. Therefore, there lacks a concrete understanding related to the potential of the energy efficiency improvement by real-time energy control strategy and the practicality of the integration of energy control module into the existing control systems.

Generally, to improve the energy efficiency of manufacturing systems with multiple machines and buffers, the following steps are designed: a) identify the opportunities of energy control for each machine; b) decide the optimal state of each machine in the system; c) make decisions and execute actions; and d) repeat steps a), b) and c) for continuous improvement. Due to the infeasibility of validating energy control models in real manufacturing lines practically, lab based experiment becomes a useful tool in analyzing the performance of those models. In this thesis, an experimental based method is proposed to study various strategies for energy efficiency improvement of complex manufacturing systems. A typical production line with multiple machines and buffers is established in both software testbed and hardware testbed, and a framework of real time energy control is also proposed and implemented in both testbeds. The **objective** of this thesis is to a) identify the potential of energy savings in modern manufacturing systems with multiple machines and buffers by energy control policy; b) examine the feasibility of the application of new energy control module under existing control systems; and c) provide a

generalized testbed framework with different functionality modules by using object-oriented programming which can be easily adjusted and fit into different systems for different research purposes.

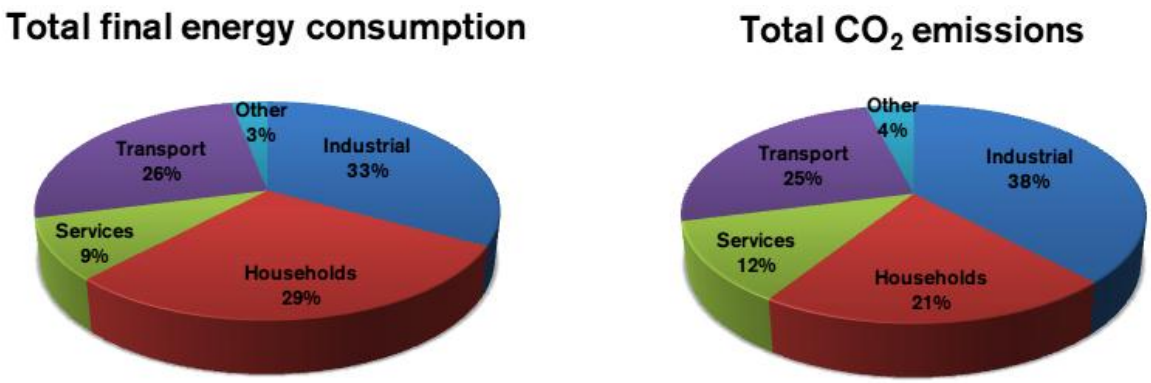
1. INTRODUCTION

In 2010, world primary energy production has increased by 145.4% compared to 1990. Fossil fuel, the largest source of energy production, accounts for about 81% of the total source supply around the world (see Figure 1) (Enerdata 2011; IEA 2011). It leads to a continuous depletion of natural resources and thus the prices of resources are irreversibly increasing. Among the worldwide energy consumption, the industrial sector is considered the largest contributor consuming about one-third of the global primary energy production and generating 38% of carbon dioxide emissions according to IEA statistics as shown in Figure 2 (IEA 2008).



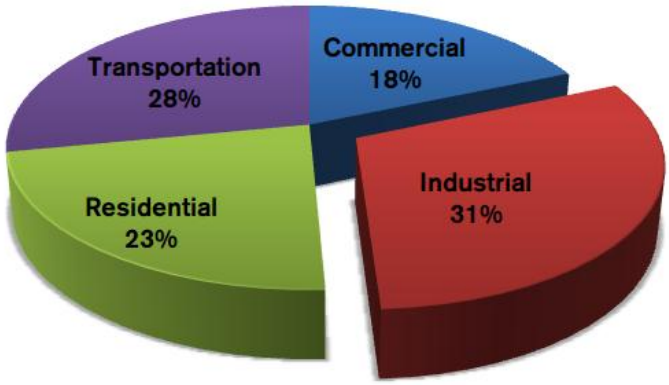
Source: (Enerdata 2011)

Figure 1 Global Primary Energy Production and Consumption, 1990-2010



Source: (IEA 2008)

Figure 2 Global Energy Consumption and CO_2 Emissions by End-use Sector, 2005

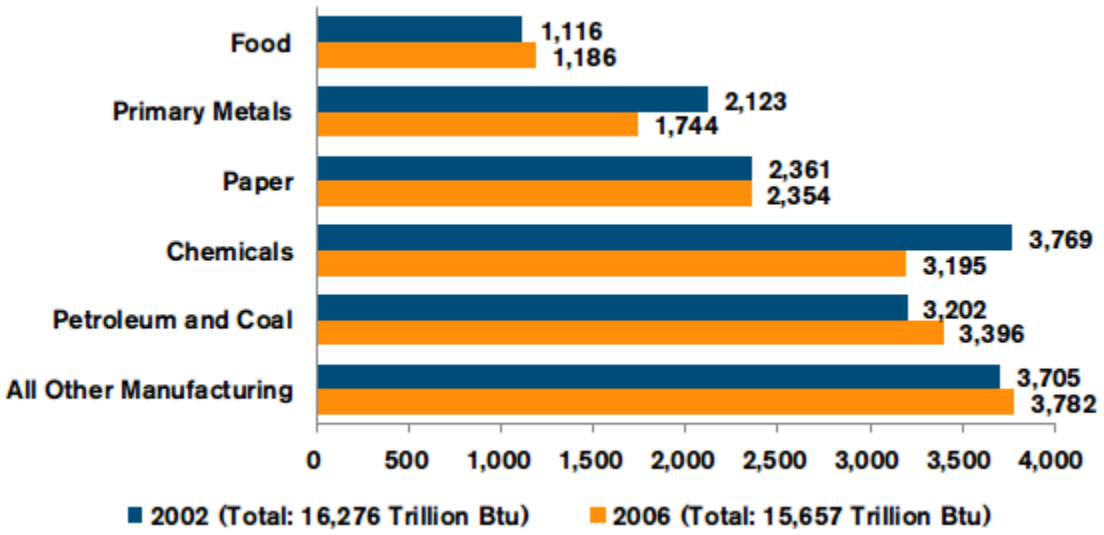


Source: (EIA 2011)

Figure 3 U.S. Energy Consumption by End-use Sector, 2010

For the United States, although its population is less than 5% of the whole population in the world, the energy consumption accounts for about 25% of worldwide volume (Park et al. 2009).

Similar to the distribution worldwide, about one-third of the energy consumed in the United States is attributed to the industrial sector (see Figure 3) (EIA 2011), among which the manufacturing is regarded as a major subsector whose energy consumption distribution is shown in Figure 4 (EIA 2006).



Source: (EIA 2006)

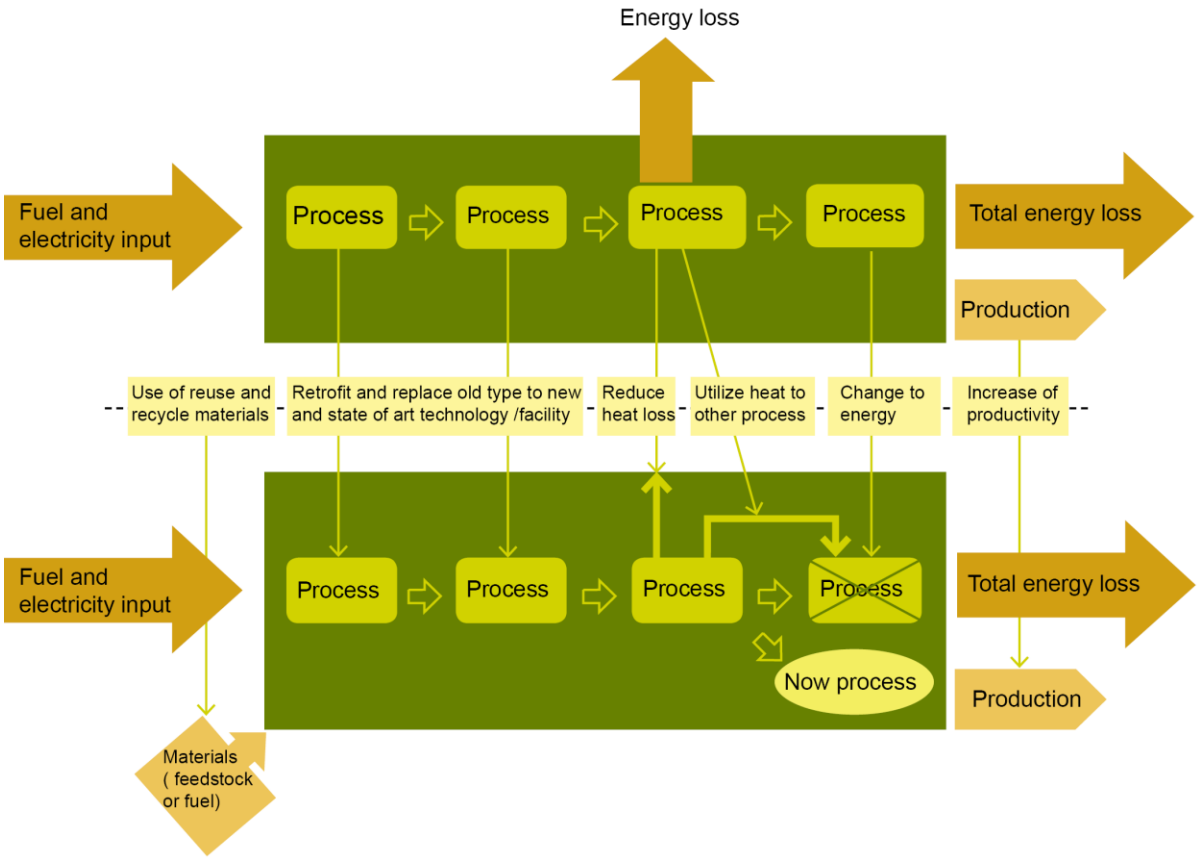
Figure 4 Manufacturing Energy Consumption Survey: Industrial Energy Uses by Sector

Traditionally, the energy consumption of manufacturing systems has not been seriously considered a primary factor for decision making, compared to other factors, *e.g.*, productivity and quality, for the sake of its relatively low contribution to the total operation cost (Galitsky and Worrell 2008). Therefore, most existing research efforts focus on the productivity analysis,

maintenance policy, quality improvement, throughput bottleneck identification, *etc.* (Lu et al. 2011; L. Li et al. 2009; L. Li, Ambani, and Ni 2009; J. Li, Meerkov, and Zhang 2010).

Recently, due to the rapid rise of energy price and increasing social pressure on the environment, more and more researchers have gradually realized the significance of reducing energy consumption of the manufacturing systems for the sake of the increasing energy demand as well as the projections for a huge shortage of fossil fuels in the foreseeable future. Generally the research on energy consumption is branching out in the following two directions: i) the discovery and development of new energy sources that do not emit or emit less greenhouse gases (GHG) and ii) the improvement of the energy efficiency under current energy modes. A great deal of effort has been made towards the new energy source techniques such as wind, solar, and tide energy. For a long term horizon, these endeavors are definitely worthwhile although many challenges are inevitable. At the same time, extensive studies that examine the energy efficiency for diverse end-use sectors of the U.S. industry have also been implemented (Worrell et al. 2009). The general consensus from these studies shows that energy efficiency improvement rate has typically been around 1% per year (Worrell et al. 2009). However, energy consumption still increases continuously from 67.84 quadrillion Btu in 1970 to approximate 99.74 quadrillion Btu in 2004 (Kankana 2008), and so the current improvement level of energy efficiency could not keep up with the increase in energy demand. Therefore, huge potentials as well as requirements exist to further improve energy efficiency and reduce energy consumption and GHG emission in a majority of industrial sectors.

Nevertheless, research with respect to the improvement of the energy efficiency of current energy modes is confined in the single machine system or specific process level (Dietmair and Verl 2009; Draganescu, Gheorghe, and Doicin 2003; Mouzon and Yildirim 2008). Although some initial analysis and exploration have been implemented from the perspective of a system level as shown in Figure 5 (Kanako 2011), the overall progress is still lagging far behind of single machine system.



Source: (Kanako, 2011)

Figure 5 Sample of Technical Methods of Energy Efficiency Improvement

The pivotal challenging obstacles are the unique characteristics of modern manufacturing systems with multiple machines and buffers, which can be summarized as follows:

Firstly, throughput has been traditionally considered the first priority for the manufacturing enterprises. Therefore, most plants are reluctant to sacrifice original throughput for an energy saving purpose. Particularly the quantitative saving potentials are not known and therefore the tradeoff between energy cost savings and economic throughput sacrifice is difficult to be identified.

Secondly, the properties and characteristics of modern manufacturing systems make the problem complicated. On one hand, high dynamics of a manufacturing system lead to non-availability of a lookup table from which the predetermined decisions can be selected; but in practice most decisions have to be made based on the real-time online data. On the other hand, high interconnection in modern manufacturing systems leads to closely interdependent machines within the system and thus the machine's operation states, *i.e.*, production, breakdown, blockage and starvation, are determined not only by the machine itself, but also by other machines and buffers. Therefore it is very difficult to map the states of machines within the system during the duration when energy control is implemented.

Actually, energy analysis of industrial facilities has indicated that energy consumption for manufacturing processes merely accounts for a small percentage of total energy consumption. For instance, the total energy demand for physical operations performed in metal-cutting could

be quite small, as little as 10% to 20% compared to the other functions required in background for operating manufacturing equipment (Dahmus and Gutowski 2004). This suggests system-level approaches for energy efficiency improvement are required to gain significant benefits. Unfortunately, to the best of our knowledge, such approaches have not yet attracted much attention due to the reasons summarized aforementioned.

The **research objective** of this thesis is to establish an experiment-based framework to identify the potentials of energy savings for a typical manufacturing system with multiple machines and buffers by implementing energy control policy under the constraint of system throughput. A general simulation-based algorithm of energy control to find the optimal power state based on the operation state of the machines within the system is developed. A software testbed is established by using ProModel®, VBA in Excel, and C# .NET WinForms application to simulate all dynamic behaviors of modern manufacturing systems, *e.g.*, mean time between failures (MTBF), mean time to repair (MTTR), system throughput, and energy consumption profile as well. The proposed algorithm is then applied to find the energy saving potentials by implementing energy control policy under the software environment. To further illustrate the huge potentials for system-level energy efficiency improvement, a hardware testbed is also established to represent a production line to perform real-time energy control, which leads to similar results as obtained in software testbed. At the same time, we also consider the flexibility of the testbed by utilizing object-oriented and modular programming and therefore the testbed can be easily adapted into different system layouts for further experimental purposes.

2. ALGORITHM AND SOFTWARE TESTBED

2.1. Motivation

A typical manufacturing system consists of multiple machines and buffers (see Figure 6). Each machine can be in multiple operation states, *i.e.*, production, breakdown, blockage or starvation; and each buffer can be also in multiple states, depending on its capacity, *i.e.*, 0, 1, 2, … , C, where C equals to the buffer capacity. Previous research showed that machine blockage/starvation times account for about 20% of the total scheduled operation time (Sun et al. 2011). However, the majority of the existing commercialized Manufacturing Execution System (MES) does not include an energy management module. Accordingly, a huge amount of energy is wasted during those idle periods due to the lack of the effective decision-making tools.

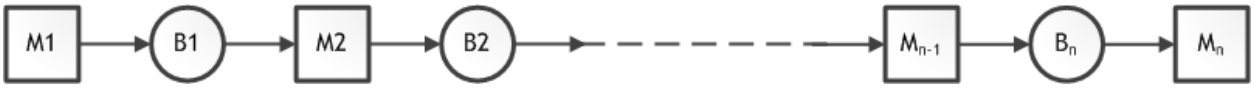


Figure 6 A Tandem Manufacturing System with n Machines and n-1 Buffers

In this research, the basic idea is to implement energy control during the idle periods of the manufacturing systems, *i.e.*, blockage or starvation. Considering the fact that the new type of motors equipped with multiple power adjustable drives becomes more and more popular, we could adjust the power level of a certain machine into a lower level when it is detected to be idle

by considering the constraint: the system throughput should not be compromised for energy control purpose. However, the analytical estimation of the opportunity window for energy control, *i.e.*, the time length of blockage/starvation, is difficult because of the complexity of manufacturing systems. In this chapter, we introduce a simulation-based algorithm for production systems with multiple machines and buffers to accurately obtain the opportunity windows for energy control and calculate the most energy efficient state for the idle machines to be adjusted in the system and thus the energy saving potentials can be obtained. To validate this algorithm, a software based testbed is established by using ProModel®, ActiveX Automation Control, VBA in Excel and C# .NET WinForms application. The statistical result of energy saving potentials is obtained by running the simulation model for fifty replications.

2.2. Model Algorithm

2.2.1. Notations

The notations are defined as follows (L. Li et al. 2012).

i	the index of the machine in system, $i = 1, 2, \ldots, n$
k	the index of replications that the simulation model run, $k = 1, 2, \ldots, K$
j_{ik}	the index of the occurrences of blockage or starvation of a machine i during the k^{th} replication, $j_{ik} = 1, 2, \ldots, J_{ik}$
M_i	the corresponding sequence of machine in system
B_i	the corresponding sequence of buffer in system

BLB_{ij}^{k} time that the j^{th} blockage of M_i begins in the k^{th} replication, $i = 1, 2, \ldots, n$, $k = 1, 2, \ldots, K$, $j = 1, 2, \ldots, J_{ik}$

BLE_{ij}^{k} time that the j^{th} blockage of M_i ends in the k^{t} replication, $i = 1, 2, \ldots, n$, $k = 1, 2, \ldots, K$, $j = 1, 2, \ldots, J_{ik}$

STB_{ij}^{k} time that the j^{th} starvation of M_i begins in the k^{th} replication, $i = 1, 2, \ldots, n$, $k = 1, 2, \ldots, K$, $j = 1, 2, \ldots, J_{ik}$

STE_{ij}^{k} time that the j^{th} starvation of M_i ends in the k^{th} replication, $i = 1, 2, \ldots, n$, $k = 1, 2, \ldots, K$, $j = 1, 2, \ldots, J_{ik}$

ΔBL_{ij}^{k} time length of the j^{th} blockage of M_i in the k^{th} replication, it equals to $BLB_{ij}^{k} - BLE_{ij}^{k}$

ΔST_{ij}^{k} time length of the j^{th} starvation of M_i in the k^{th} replication, it equals to $STB_{ij}^{k} - STE_{ij}^{k}$

ΔI_{ij}^{k} represents either ΔBL_{ij}^{k} or ΔST_{ij}^{k}

P_i^f power level of M_i when M_i is ready to produce

P_{ij}^{q} power level q of the j^{th} blockage or starvation for M_i, $q \in Q_i$

Q_i set of different power levels for M_i

T_{ij}^{fq} transition time of the j^{th} blockage or starvation for M_i from P_i^f to P_{ij}^{q}

T_{ij}^{qf} transition time of the j^{th} blockage or starvation for M_i from P_{ij}^{q} to P_i^f

P_{ij}^{fq} average power per time unit of the j^{th} blockage or starvation for M_i during T_{ij}^{fq}

P_{ij}^{qf} average power per time unit of the j^{th} blockage or starvation for M_i during T_{ij}^{qf}

E_{ij}^{fq} transition energy of the j^{th} blockage or starvation for M_i from P_i^f to P_{ij}^{q}, it equals to $P_{ij}^{fq} \cdot T_{ij}^{fq}$

E_{ij}^{qf} transition energy of the j^{th} blockage or starvation for M_i from P_{ij}^{q} to P_i^f, it equals to $P_{ij}^{qf} \cdot T_{ij}^{qf}$

E_a^k total energy consumption during idle period when power adjustment policy is implemented in replication *k*

E^k total energy consumption without executing the power adjustment policy in replication *k*

TP throughput of the system

p^k ratio of energy saving in replication *k*

2.2.2. Selection of Optimal Energy Level

In order to perform energy adjustment during a machine idle (blockage or starvation) period without influencing system throughput, the baseline model without adjusting machine power level when the machine is detected to be blocked or starved, is initially run to record the runtime parameters such as BLB_{ij}^k , BLE_{ij}^k , STB_{ij}^k , STE_{ij}^k , *TP*, and energy consumption. After that, a control module including VBA in Excel and C# .NET WinForms application (see details in Chapter 2.3) is called to check the feasibility and profitability of different objective power levels by using (2.1) and (2.2) to see if the time length is long enough to perform adjustment and the energy saving can be achieved.

$$\Delta I_{ij}^k > T_{ij}^{fq} + T_{ij}^{qf} \tag{2.1}$$

$$P_i^f \cdot \Delta I_{ij}^k > P_{ij}^q \cdot \left(\Delta I_{ij}^k - T_{ij}^{fq} - T_{ij}^{qf} \right) + E_{ij}^{fq} + E_{ij}^{qf} \tag{2.2}$$

The qualitative analysis of feasibility check and profitability check according to (2.1) and (2.2) is shown in Figure 7.

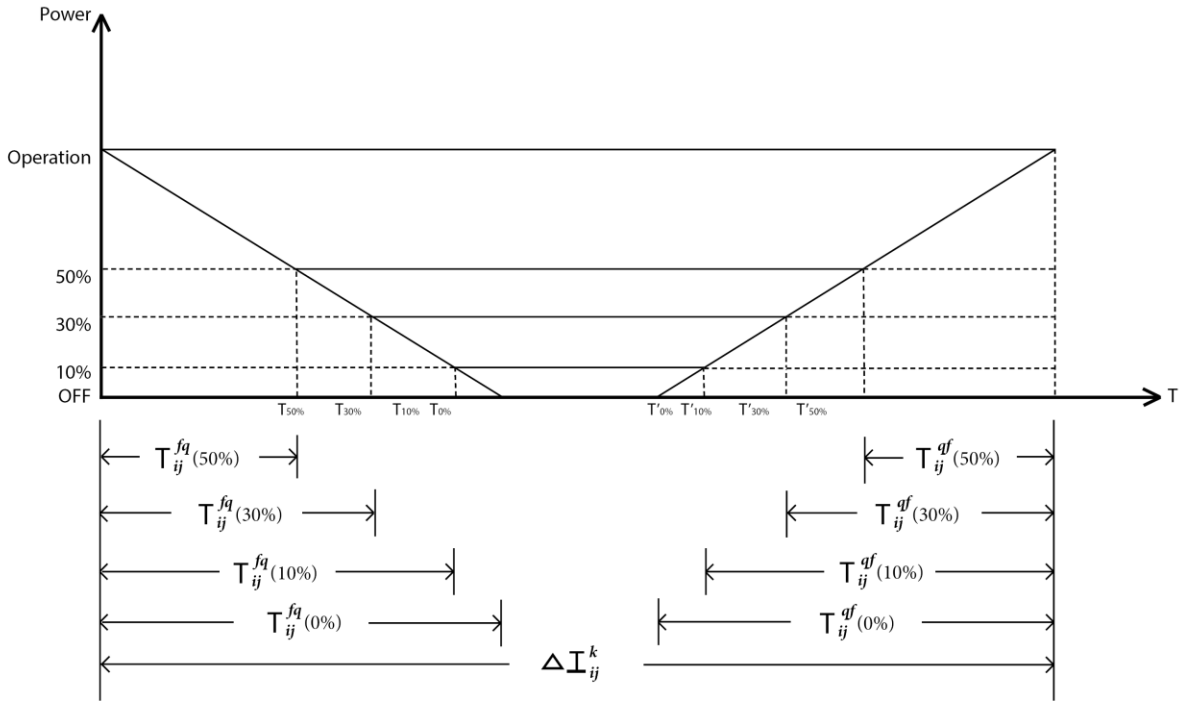


Figure 7 Feasibility Check and Profitability Check for Power Adjustment

The horizontal axis represents the time length of the j^{th} blockage/starvation for M_i and the vertical axis represents the different power level of M_i. $T_{ij}^{fq}(50\%)$, $T_{ij}^{fq}(30\%)$, and $T_{ij}^{fq}(10\%)$ represent the time length to adjust power level from P_i^f to a specific power level P_{ij}^q, that is, 50%, 30%, and 10% of P_i^f for M_i in j^{th} blockage or starvation.

If both (2.1) and (2.2) are satisfied, then the optimal power level $P_{ij}^{q^*}$ and relevant parameters can be identified by (2.3). Otherwise, keep the machine with the original power level P_i^f. The flow chart describing the algorithm for the selection of an optimal energy level is shown in Figure 8.

$$P_{ij}^{q^*} = \arg\min_{P_{ij}^{q^*}}[\left(\Delta I_{ij} - T_{ij}^{fq} - T_{ij}^{qf}\right)\cdot P_{ij}^{q} + T_{ij}^{fq}\cdot P_{ij}^{fq} + T_{ij}^{qf}\cdot P_{ij}^{qf}] \tag{2.3}$$

2.2.3. Energy Saving Potentials

The total energy consumption during idle period when power adjustment policy is implemented in replication *k* can be described as follows:

$$E_a^k = \sum_i \sum_j [(\Delta I_{ij} - T_{ij}^{fq^*} - T_{ij}^{q^*f})\cdot P_{ij}^{q^*} + T_{ij}^{fq^*}\cdot P_{ij}^{fq^*} + T_{ij}^{q^*f}\cdot P_{ij}^{q^*f}] \tag{2.4}$$

So the ratio of energy saving for k^{th} replication can be obtained by (2.5).

$$p^k = 1 - \frac{E_a^k}{E^k} \tag{2.5}$$

Therefore the average energy saving potentials can be obtained by (2.6),

$$\overline{p} = 1 - \frac{\overline{E_a}}{\overline{E}} \tag{2.6}$$

where $\overline{E_a} = \frac{\sum_{k=1}^{K} E_a^k}{K}$ and $\overline{E} = \frac{\sum_{k=1}^{K} E^k}{K}$.

The 95% confidence interval can be obtained by (2.7),

$$\begin{cases} C.I.(E) = \overline{E} \pm t_{\frac{\alpha}{2}} \cdot \frac{S_E}{\sqrt{K}} \\ C.I.(E_a) = \overline{E_a} \pm t_{\frac{\alpha}{2}} \cdot \frac{S_{E_a}}{\sqrt{K}} \\ C.I.(p) = \overline{p} \pm t_{\frac{\alpha}{2}} \cdot \frac{S_p}{\sqrt{K}} \end{cases} \tag{2.7}$$

where *C.I.(E)*, *C.I.(E_a)*, and *C.I.(p)* represent the confidence interval of *E*, E_a, and *p*, respectively. S_E, S_{E_a}, and S_p are the sample standard deviation of *E*, E_a, and *p* obtained through *K* replications, respectively. $t_{\frac{\alpha}{2}}$ represents the critical value for the Student's t-distribution under the α level of confidence.

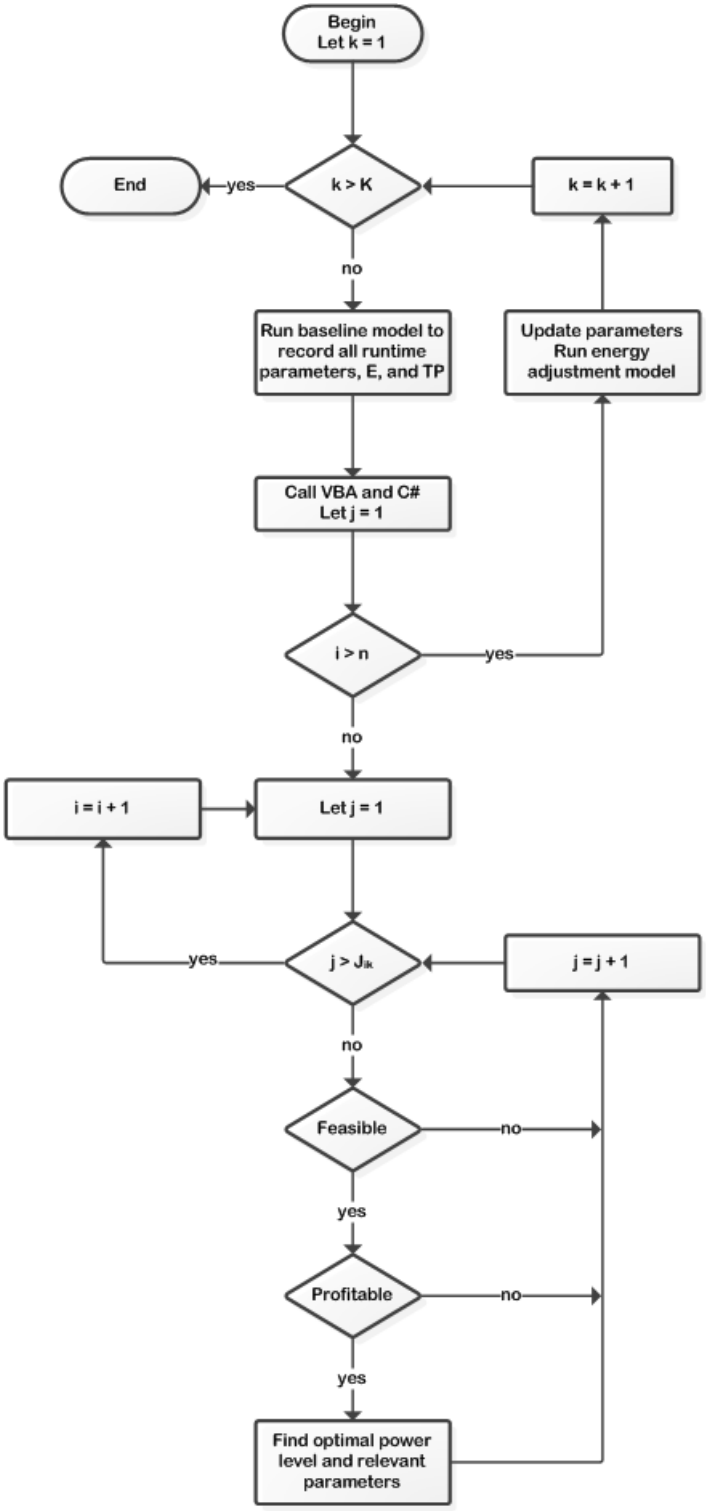


Figure 8 Flow Chart of the Simulation Based Procedure

2.3. Software Testbed

To apply the algorithm described in Chapter 2.2, a software testbed is established by mainly using ProModel®, a discrete event simulation software used for evaluating, planning or designing manufacturing, warehousing, logistics and other operational and strategic situations. The manufacturing system is established in the software by defining machine locations, process entities, sequential relations, and other parameters that describe system behaviors like MTBF, MTTR, system throughput, and power consumption. Figure 9 illustrates a snapshot of the manufacturing line layout created in ProModel®.

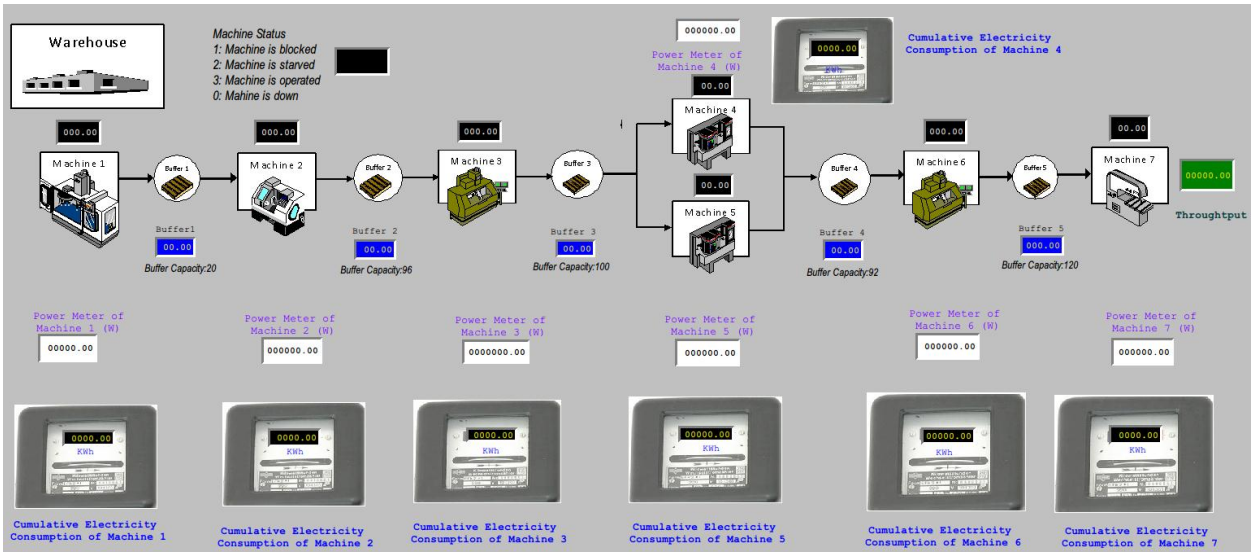


Figure 9 Layout of Simulation Model

To realize the edition of large pieces of reusable codes by using ProModel®, ActiveX Automation Control is used to build customized user interfaces by ActiveX-enabled language. The functionalities including adding, revising, deleting model data, control ProModel®, and extracting output data can be realized outside of ProModel®. In addition, VBA based Excel 2010 spreadsheet (see Figure 10) is developed to handle relevant logic codes, processing codes and runtime action variables. The corresponding records are extracted and the relevant parameters are calculated and fed back into the simulation so that the energy adjustment scenario can be performed with the same machine reliability condition as the baseline.

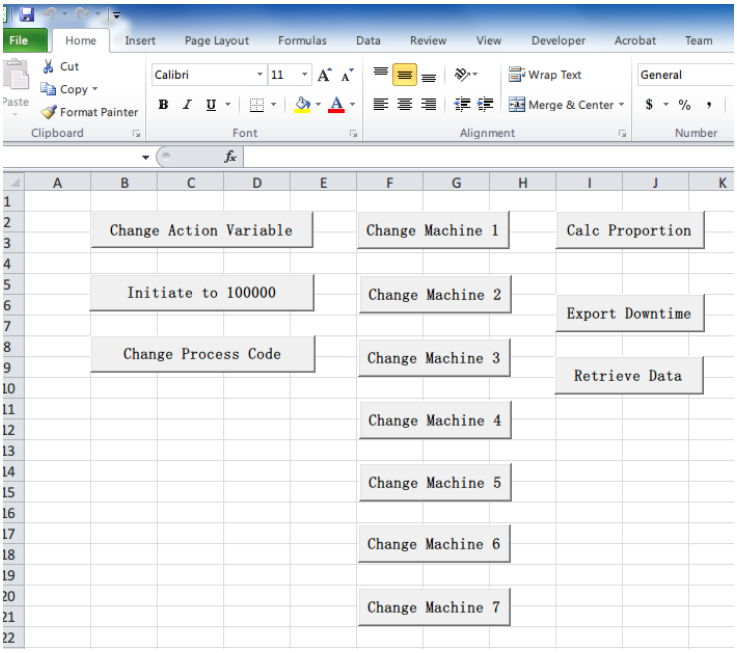


Figure 10 VBA Control Tool Based on Excel 2010

At the same time, C# .NET Windows Forms application (see Figure 11) is developed to generate the blockage and starvation runtime action codes for machines in the simulation model. Relevant parameters for different machines can be easily modified for different replications and thus the statistical results can be obtained easily.

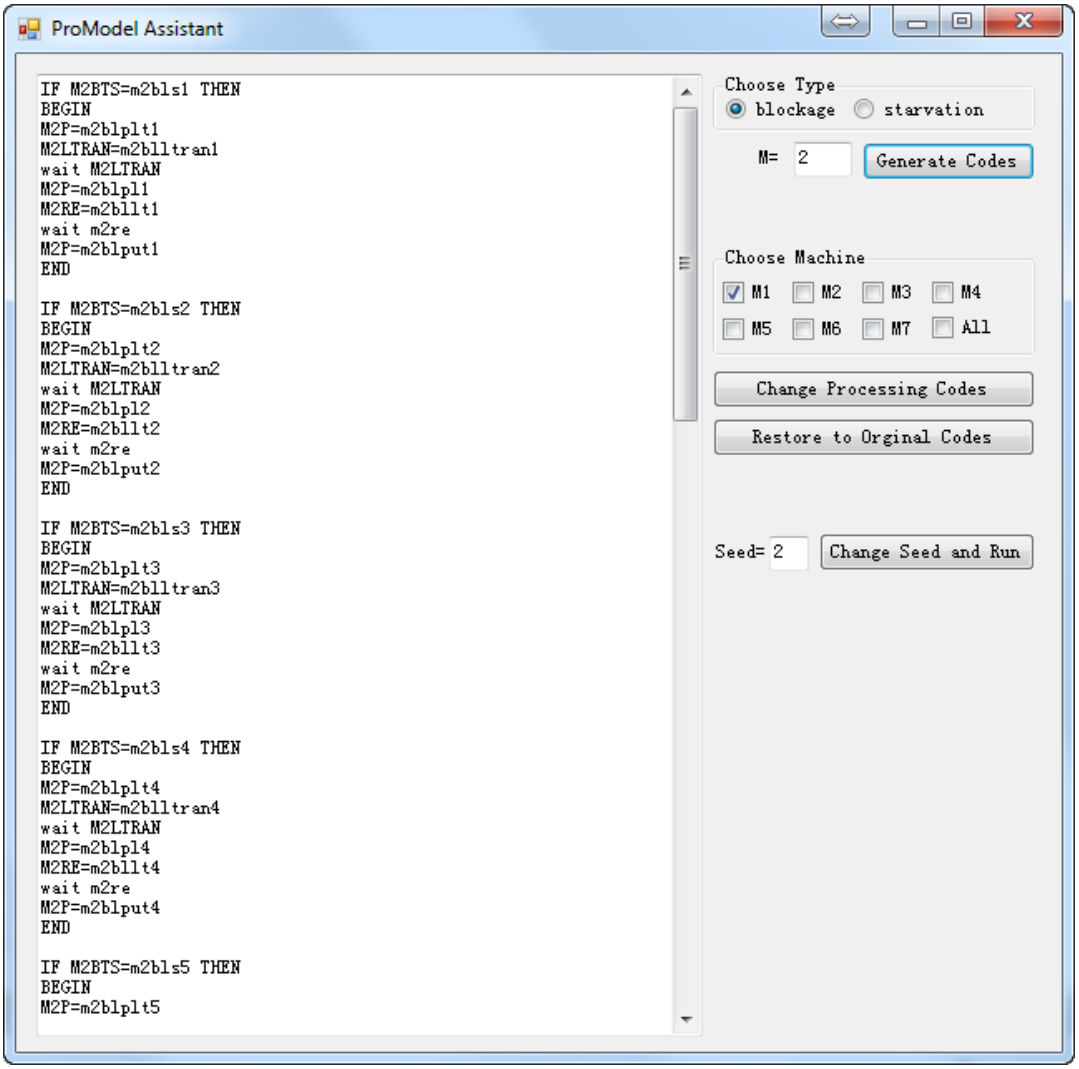


Figure 11 ProModel Assistant Designed in C# .NET

2.4. Case Study

In order to validate the algorithm illustrated in Chapter 2.2 and realize the functionality of the software testbed described in Chapter 2.3, a section of an automotive production line with seven machines and five buffers is considered as shown in Figure 12.

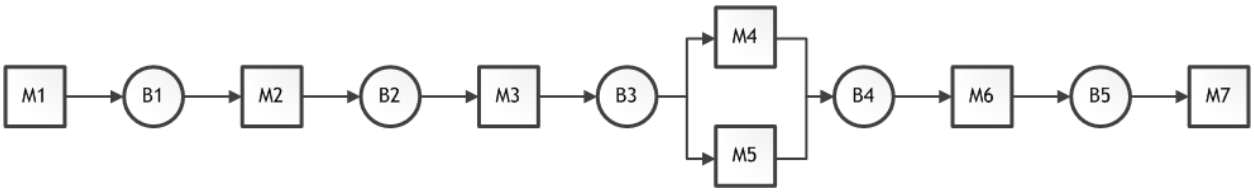


Figure 12 A Hybrid Production Line with Seven Machines and Five Buffers

Real data from the plant, *i.e.*, mean time between failures (MTBF), mean time to repair (MTTR), and machining cycle time of the seven machines are listed in TABLE I. The power consumption during operation and warm up time for each machine are also assumed. Buffer capacity and initial buffer contents of the five buffers are listed in TABLE II. Apart from that, we consider the discrete power adjustment configurations when computing the optimal power level. TABLE III shows five discrete adjustable power levels in our case.

TABLE I
BASIC SETTINGS FOR MACHINES IN SOFTWARE TESTBED

	MTBF (min)	MTTR (min)	Power (kW)	Cycle Time (min)	Warm up Time (min)
M_1	85	10.34	7.3	0.5	0.8
M_2	432.9	4.95	7.3	0.445	0.8
M_3	34.9	7.27	5	0.447	0.55
M_4	17.8	6.81	7.5	0.96	0.82
M_5	13.3	7.7	7.6	0.975	0.83
M_6	79.8	10.49	7.6	0.44	0.83
M_7	50.9	7.96	7.31	0.5	0.8

TABLE II
BUFFER CAPACITY AND INITIAL CONTENTS IN SOFTWARE TESTBED

	Buffer 1	Buffer 2	Buffer 3	Buffer 4	Buffer 5
Capacity	20	96	100	92	120
Initial Contents	8	44	46	44	54

TABLE III
ENERGY CONSUMPTION STATE FOR EACH MACHINE IN SOFTWARE TESTBED

Energy Consumption State	Production	Shallow Sleep	Medium Sleep	Deep Sleep	Off
Power Level	100%	50%	30%	10%	0%

Fifty replications of both the baseline and adjusted scenario are performed in this case study. For each pair of the baseline and adjusted scenario, the system reliability parameters are kept the same strictly. Figure 13 shows the machine status distribution for one certain replication.

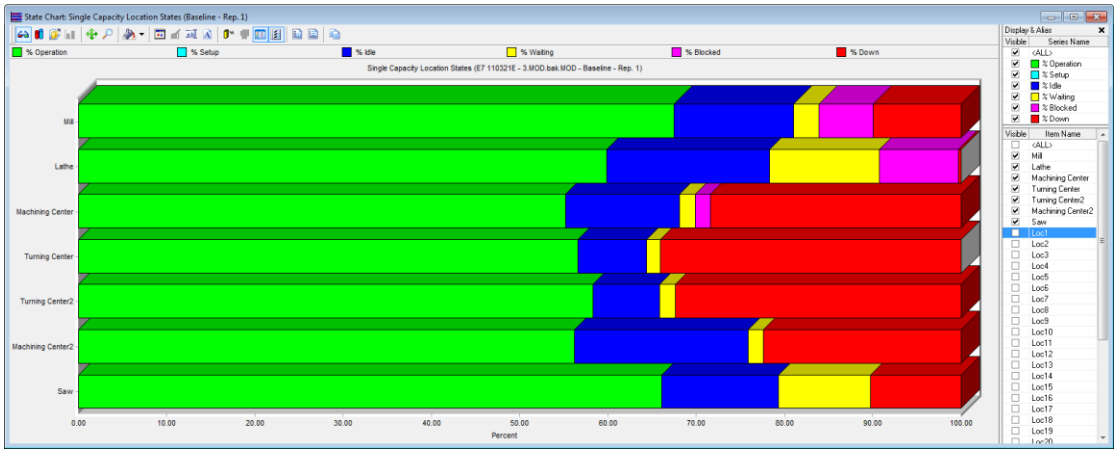


Figure 13 Results of Machine State for a Certain Replication

The actual average machine status after 50 replications of simulation are captured and shown in TABLE IV. It can be observed that in an 8-hour shift, there is approximate 21% of the time during which energy is wasted without production in the system.

TABLE IV
TIME LENGTH OF EACH STATE OF MACHINE IN SOFTWARE TESTBED

	Scheduled Operation Time (min)	Unscheduled Downtime (min)	Blockage/Starvation (min)	Time Percentage of Energy Waste
M1	480	70.6	41	10%
M2	480	4.6	156	33%
M3	480	79.7	100	25%
M4	480	116.5	22	6%
M5	480	171.7	15	5%
M6	480	53.0	94	22%
M7	480	51.4	111	25%
Total	3360	547.7	539	Average: 21%

The energy consumption is compared between the baseline model and the adjustment scenario. It can be observed from TABLE V that approximate 13.8% of the total energy can be conserved by adjusting machines' power level when machines are blocked or starved according to the method addressed in Chapter 2.2. The benefits will be proportional to the seriousness of the blockage and starvation of the system, particularly for those machines with higher P_i^f .

The throughput is also compared. It can be observed from TABLE VI that no throughput loss occurs when a power state adjustment is executed compared to the baseline scenario.

TABLE V
COMPARISON OF ENERGY CONSUMPTION BETWEEN BASELINE AND POWER ADJUSTMENT MODEL IN SOFTWARE TESTBED

	Original Energy Consumption	Energy Consumption with Power Adjustment	Energy Consumption Saving
Energy Consumption (kWh)	335	288	13.8%
95% Confidence Interval	(331, 339)	(282, 294)	(11.3%, 16.3%)

TABLE VI
COMPARISON OF THROUGHPUT BETWEEN BASELINE AND POWER ADJUSTMENT MODEL IN SOFTWARE TESTBED

	Original Throughput	Throughput after Power Adjustment
Throughput	636	634
95% Confidence Interval	（622, 650）	（618, 650）

2.5. Conclusion

In this chapter, a simulation-based method to find the optimal power state for the blockage/starvation machine in the manufacturing systems is proposed and a general software testbed is established under the ProModel® framework. A case study is implemented in the established testbed with the proposed method. Averagely, approximate 13.8% of energy saving is realized without impacting the system throughput. Furthermore, the functionality of the ProModel® is strengthened by jointly utilizing VBA based Excel 2010 spreadsheet, ActiveX Automation Control, and C# .NET WinForms application and so the further revision of the testbed can be easily implemented for other different research purposes.

3. HARDWARE TESTBED FOR ENERGY CONTROL

3.1. Motivation and General Description

The software testbed established in Chapter 2 can accurately simulate the reliability behaviors of manufacturing systems if the relevant data used in the model are from real world. However, the energy consumption profile cannot be represented as accurately as reliability behaviors in software environment even though real energy related data is available due to the functionality limitation of software, *e.g.*, energy measurement and power fluctuation. After all, simulation is an ideal model. To make the results of our algorithm from software testbed more convincible and closer to real industrial environment, a hardware testbed using Direct Current (DC) motors and virtual buffers is established in this chapter. It can be treated as a bridge between the theoretical simulation models and the practical plant applications in energy control area. The effectiveness of the software testbed can be further validated by testing the same cases through the hardware testbed.

Generally, the hardware testbed described in this chapter includes both hardware and programming platform. The hardware infrastructure consists of both mechanical and electrical resources. The programming platform is developed by three-tier client–server architecture. The design of the hardware testbed considers the flexibility. The object-oriented programming and parameterization of the programming platform secure the code reusability and simplicity of modification for layout rearrangements. It is convenient to fit for the validation of different

manufacturing lines and different research related to system-level energy and throughput analysis can be performed.

3.2. Hardware Testbed Hierarchy

The first step to develop a hardware testbed is to determine the general hierarchy. In our task, the hierarchy is designed in Figure 14. There exist three levels, *i.e.*, programming platform, logic control, and hardware infrastructure.

The lowest level functionality is hardware infrastructure consisting of mechanical executive components (seven DC motors) and electrical control modules (seven Jaguar controllers from Texas Instrument Inc.). Motors are controlled by the Jaguar controllers and the Jaguar controllers are communicated with the logic control level which plays an important role as it coordinates the discrete dynamics between the hardware infrastructure and programming platform. All machines and buffers are coordinated separately within logic control level. Machines are coordinated by both hardware communication and core processing unit, while buffers are only coordinated by core processing unit since there is not any real hardware to represent buffers. To perform complex analysis and to make decisions of runtime machine behaviors involving data and complex logic, programming platform is required to integrate system events and the information between different functional modules since logic control level cannot handle the whole system functionality of the hardware testbed alone.

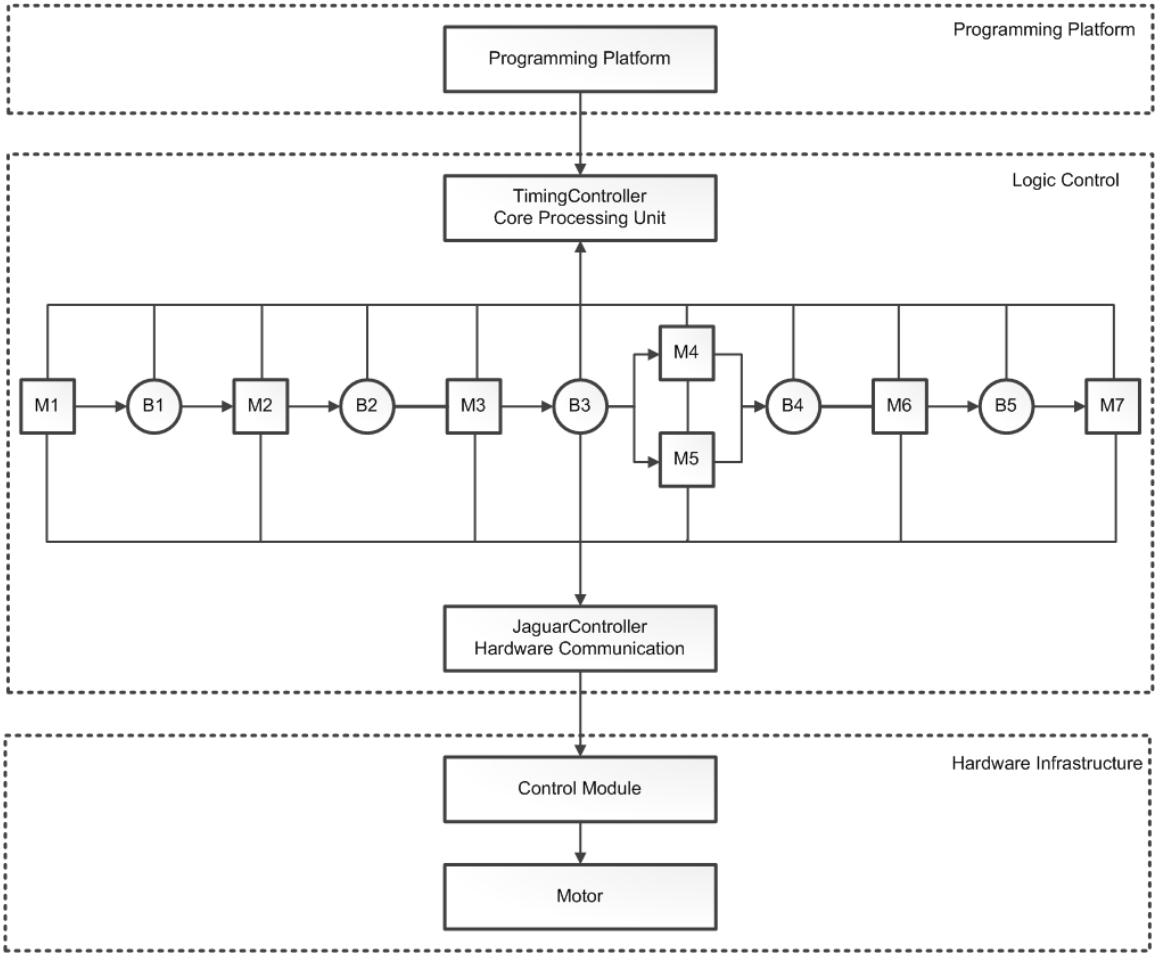


Figure 14 Hardware Testbed Hierarchy

3.3. Hardware Infrastructure

After identifying the overall hierarchy, we focus on the hardware realization. The design of the hardware architecture is illustrated in Figure 15. Two power supplies support the whole system and the red lines show the power output distribution. Seven motors equipped with two types of chucks are used to represent seven machines. They are connected to the Jaguar controllers

directly through the black lines. The blue lines show the CAN network of Jaguar controllers connection to PC, which also means the connection to the medium level of logic control as explained in Chapter 3.2.

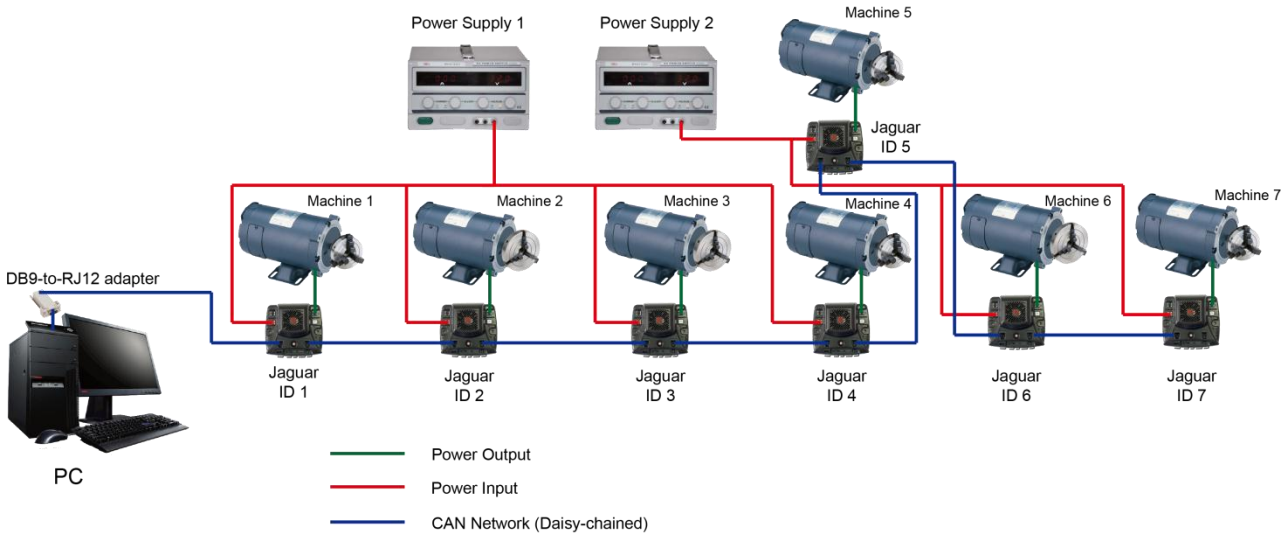
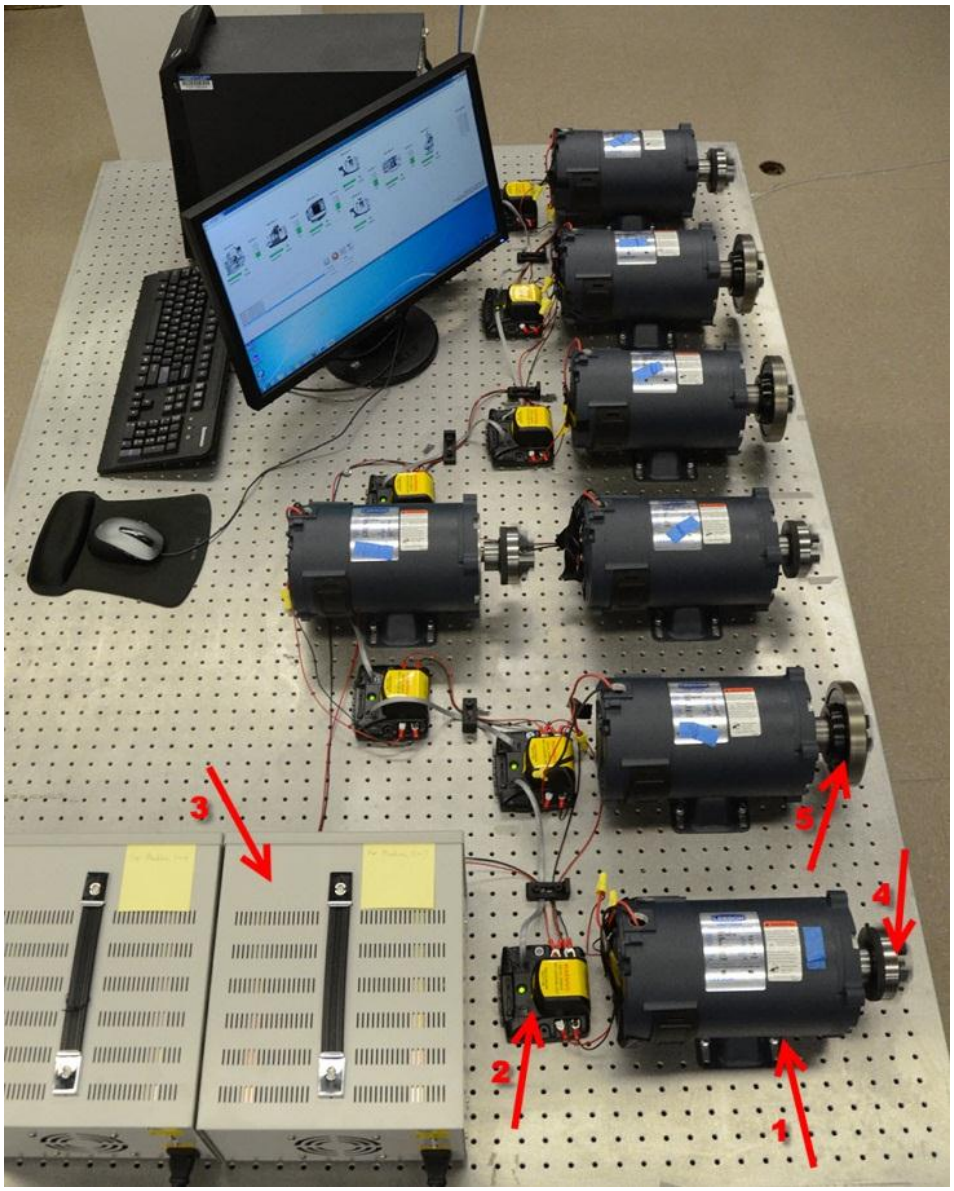


Figure 15 Hardware Physical Connection

3.3.1. Mechanical Resources

The hardware infrastructure deployment is shown in Figure 16. Different components of the hardware are illustrated as below.



[1]LEESON 1/2HP 1800RPM 56C Frame 24V DC TENV.

[2]Texas Instruments Stellaris® Brushed DC Motor Control Module with CAN (MDL-BDC24).

[3]Mastech Variable Regulated DC Power Supply 0-30V 0-20A.

[4]Grizzly Industrial®, Inc. 3" 3-Jaw Wood Chuck - 5/8" Unthreaded.

[5]Grizzly Industrial®, Inc. 5" 3-Jaw Wood Chuck - 5/8" Unthreaded.

Figure 16 Hardware Infrastructure Deployment

Motor

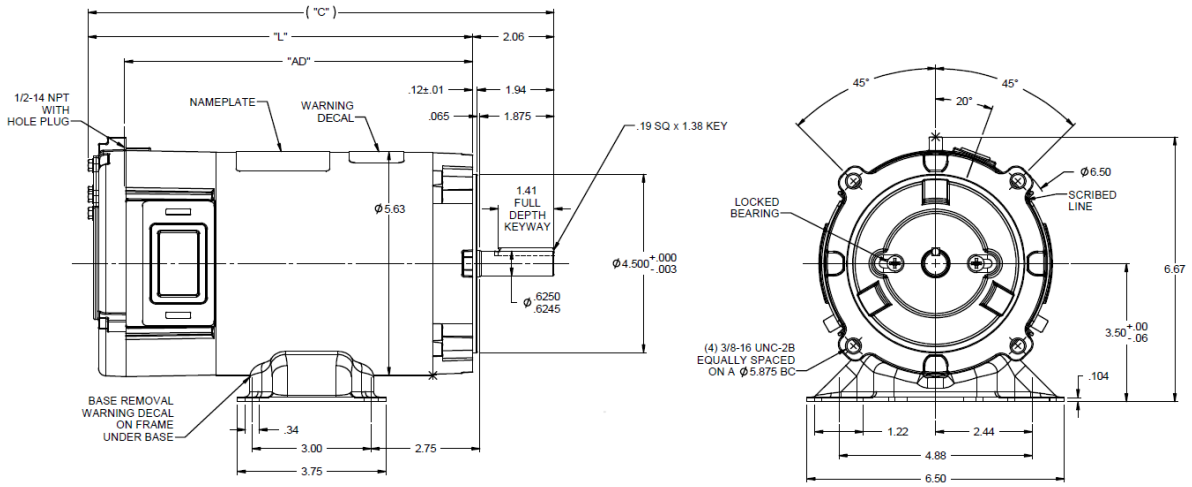
Low voltage permanent magnet DC motors are suitable for installations with lab-equipped DC power supply. Since the testbed is required to run 8-hours shifts over and over again, larger over-sized brushes assuring longer brush life are necessary. The holes of the mounting flanges match the pitch of the lab table. Four Medium-Strength Steel Cap Screws fasten the motors, as shown in Figure 17, in case of the motor shocks.

Figure 17 Motor Fastening

Chuck

To capture the motor energy in a real-time system with load, we install a heavy chuck on the motor shaft to represent a lathe or machining center. With the heavy load, we can measure the

voltage and current of the motor when it is in operation. According to the drawing of the motor shown in Figure 18 (LEESON 2010), we select two types of unthreaded micro 3-Jaw wood chucks with bore diameter of Φ 5/8", which fits the motor shaft diameter of Φ 5/8". Four 3" and three 5" chucks are installed on the seven motors as footnoted in Figure 16.



Source: (LEESON 2010)

Figure 18 Drawing of the Motor

Soft-Tip Set Screw

Because of the high rotating speed of the motor, the unthreaded hole without any fastening pieces may result in security concerns. However, the chucks do not have any key groove to match the 0.19SQ × 1.38 KEY. Therefore we find a unique set of screws with brass tips, which is

shown in Figure 19. The tip diameter of 4.8mm matches the 0.19SQ KEY, which is 4.826mm. Therefore the engineering tolerance/fitting is transition fits, and the chuck can be fastened.

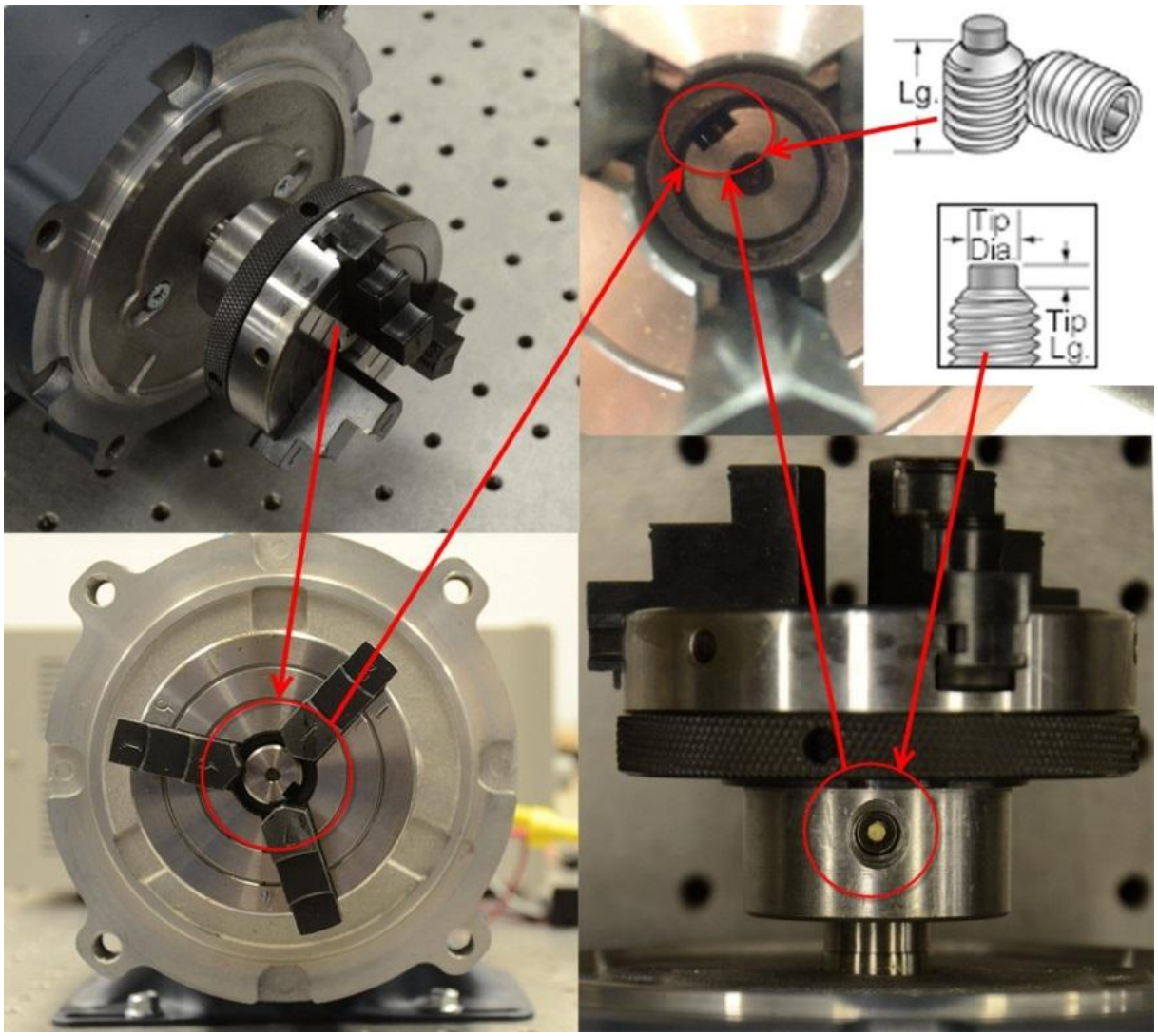


Figure 19 Chuck Fastening with Soft-tip Set Screws

3.3.2. Electrical Resources

Motor Control Module

In order to set the motor to a certain power level, a motor control module is applied to adjust the output voltage to the motor. We use Texas Instruments Stellaris® Brushed DC Motor Control Module with CAN (MDL-BDC24), also named as Jaguar. The Jaguar provides variable speed control for both 12 V and 24 V brushed DC motors at up to 40 A of continuous current, while adding a new RS232-based serial control input that also functions as a serial-to-CAN bridge. Following this way, we can use the PC COM port to control the Jaguar, instead of a microcontroller unit. Apart from that, Jaguar supports the simultaneous use of CAN for monitoring voltage and current, therefore no additional test instruments are required. Figure 20 shows the sixth Jaguar connecting to the sixth motor.

The Jaguar is fixed by a unique ribbed plastic anchors with a pan head combo drive screw (see Figure 21).

Figure 20 Jaguar Motor Control Module

Figure 21 Jaguar Fixation

Power Supply

The selection of the power supply should consider the peak current. After testing the current of each motor, the maximum value is 3.9 A, so seven machines require a total maximum current of:

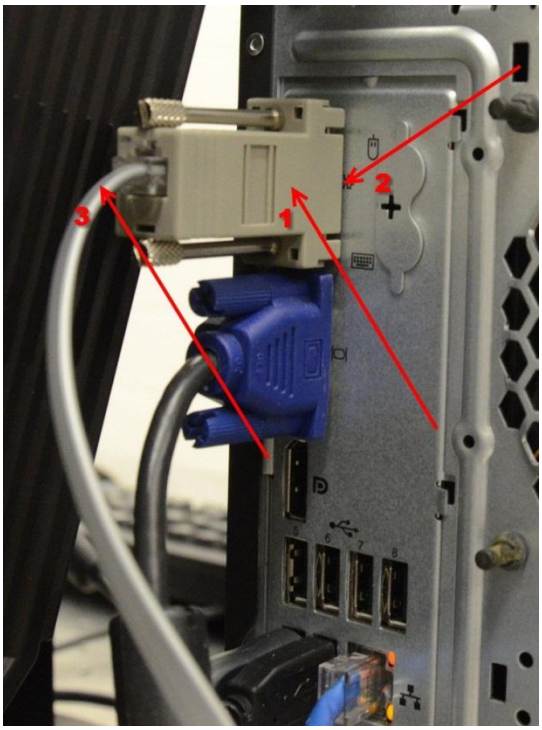
$$3.9 \times 7 = 27.3A \tag{3.1}$$

We installed two Mastech Variable Regulated DC Power Supplies with a maximum of a 20 A current within 30 V voltage output respectively in parallel (see Figure 15 and Figure 16). Power supply 1 supports machine 1 through machine 4, while power supply 2 supports machine 5 through machine 7, respectively.

3.3.3. Connection between Mechanical and Electrical Resources

Connect Jaguar to PC

Connect the DB9-to-RJ12 adapter to the COM port of PC (TI 2012a). Then connect one end of the 6P6C cable to the adapter and the other end to the Jaguar's CAN/RS232 connector (left of the LED) as shown in Figure 22.



[1]DB9-to-RJ12 adapter

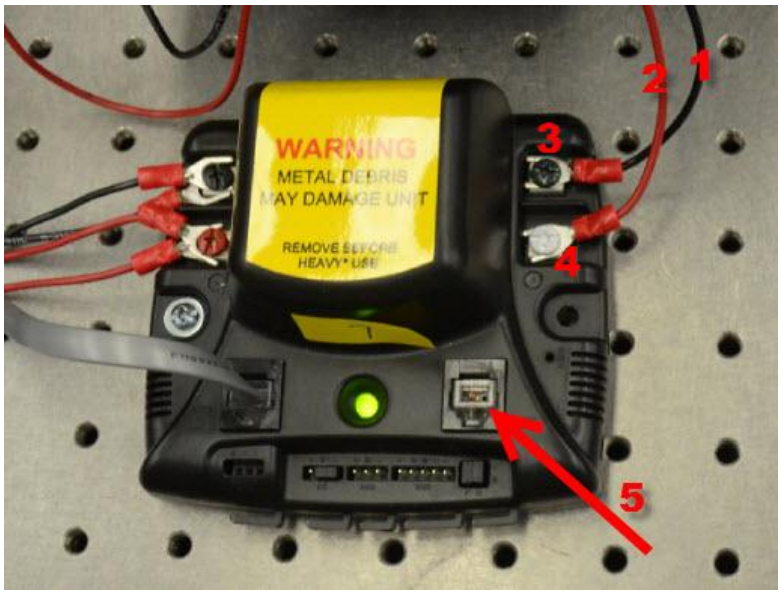
[2]PC COM port

[3]6P6C Cable

Figure 22 Connect Jaguar to PC

Connect Motor to Jaguar

Connect the black lead of the motor to the green screw terminal of the Jaguar. Then connect the red lead of the motor to the white screw terminal of the Jaguar (see Figure 23).



[1]Black lead of the motor

[2]Red lead of the motor

[3]Green screw terminal of the Jaguar

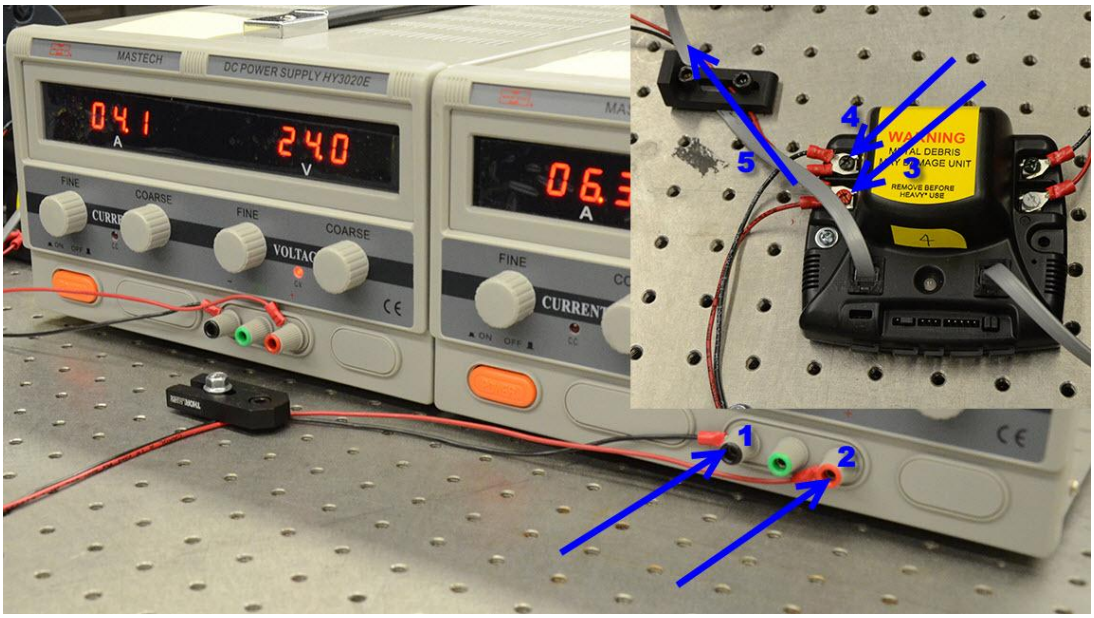
[4]White screw terminal of the Jaguar

[5]100Ω terminal resistor

Figure 23 Connect Motor to Jaguar

Connect Power Supply to Jaguar

Connect the black lead of the power supply output to the black screw terminal of the Jaguar. Then connect the red lead of the power supply output to the red screw terminal of the Jaguar. In the meantime, the black and red screws are also parallel connecting to other Jaguars for power supply (see Figure 24).



[1]Black lead of the power supply output

[2]Red lead of the power supply output

[3]Red screw terminal of the Jaguar

[4]Black screw terminal of the Jaguar

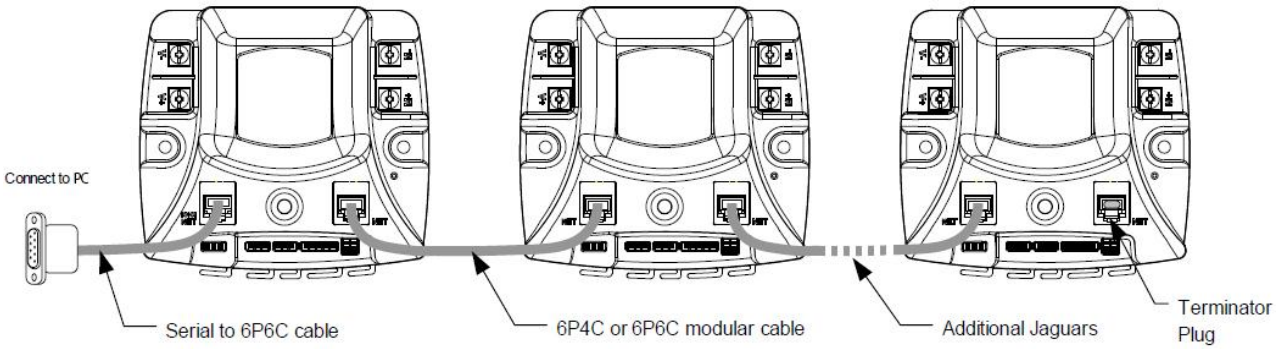
[5]Parallel connect the power supply to other Jaguars.

Figure 24 Connect Power Supply to Jaguar

Daisy Chain of Jaguar

Controller Area Network (CAN) provides a powerful interface for controlling more than one Jaguar modules. The 6P6C socket can be used for daisy-chaining with standard cables. At the end of the CAN network, which is the Jaguar for Machine 7, it should be terminated with a 100Ω

resistor as shown in the fifth footnote of Figure 23. Each Jaguar on the CAN bus is accessed using an assigned ID number. We designate Jaguar ID 1 through Jaguar ID 7 to control Machine 1 through Machine 7, as labeled and shown in Figure 15 and Figure 16, correspondingly. The CAN network topology for Jaguar daisy-chaining connection is shown in Figure 25.



Source: (TI 2012b)

Figure 25 CAN Network Topology

3.4. Programming Platform

Compared to the realization of hardware, the design of the programming platform is much more complicated so that it becomes the most challenging obstacle during the development of the hardware testbed. In this section, we elaborate the design process of the programming platform in detail.

3.4.1. Programming Architecture

At the beginning, the programming architecture should be defined prior to everything else. The three-tier client–server architecture including presentation tier, business logic tier, and data tier is selected. The significance of choosing this architecture is that by breaking up the programming platform into three tiers, we only have to modify a specific tier, rather than have to rewrite the entire application over (Wikipedia contributors 2012b). The parametric configuration for different kinds of manufacturing lines is possible under this design. The programming architecture is designed as shown in Figure 26.

The presentation tier consists of the main user interface which is a WinForms GUI Components designed in C# .NET. This tier is used mainly for the interaction between the internal data, complex logic, and the external user interface.

The business logic tier is comprised of two primary units, core functional unit and hardware communication functional unit. The core unit coordinates the whole system by performing complex analysis and determining the decisions of runtime machine behaviors. It also communicates with 1) hardware unit to set the corresponding motor executive commands or get the real-time data from the motor, 2) data tier to retrieve or coordinate runtime data, and 3) presentation tier to present all the runtime information including machine energy status, buffer contents, system throughput, and system behaviors. The hardware unit communicates with hardware infrastructure of seven Jaguar controllers.

The data tier consists of configuration access and data access. For configuration access, there are three parts of configuration involving machine parameters, buffer parameters and system parameters. The data can be retrieved at the very beginning before the system is implemented. During the runtime of the system, the system behavior data like the beginning and ending time of blockage or starvation and real-time energy related data from the hardware can be stored. The Excel database can be accessed and exported at the end with all the runtime data obtained and analyzed from the core functional unit.

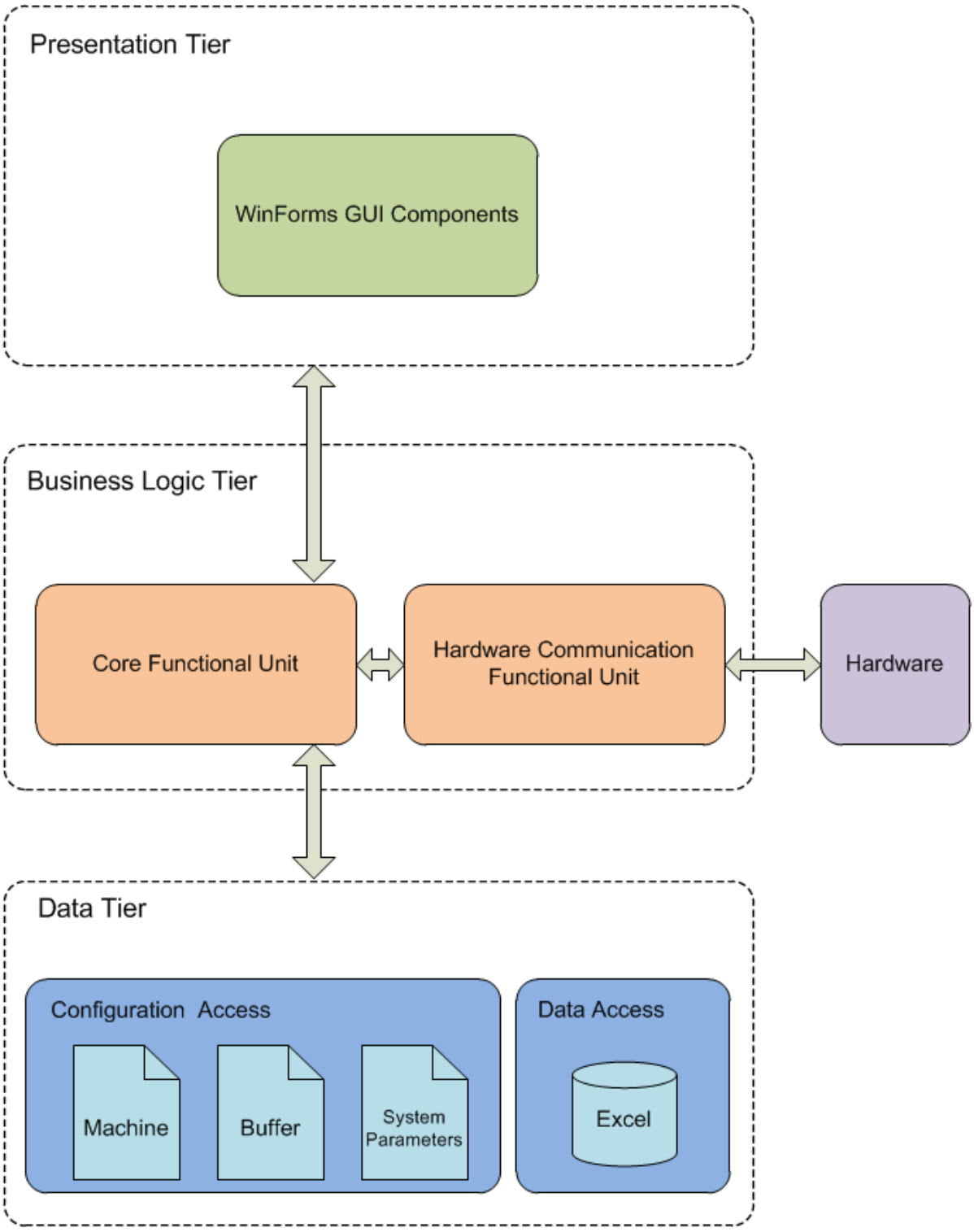


Figure 26 Programming Architecture

3.4.2. Develop Environment Considering Design Pattern

After determining the system hierarchy, we need to identify the programming environment and the language we use. On one hand, from the perspective of motor controlling, we do the analysis as follows.

Since the chip of the Jaguar controller is based on ARM (ARM is a 32-bit reduced instruction set computer) architecture and all other microcontrollers from Texas Instrument (TI) are based on ARM, so the official solution of motor controlling to realize the algorithm of Chapter 2 is to run the energy control model in an ARM embedded development environment in C++. However, due to the low clock frequency (50 MHz) and the architecture of this ARM MCU, it is not designed for large-scale scientific computing. So the official solution will not be a good choice.

At the same time, considering the fact that TI provides an application program, bdc-comm, and relevant C++ source code to control the motor in either GUI mode or command line mode, we consider modifying and rebuilding this application program to fit our model by extracting the core voltage controlling API from the bdc-comm and translating it into a Component Object Model (COM) interface so that other high level language can easily call the API to realize the motor control. However, the workload of this idea is too heavy because 1) large capacity of code is required to be understood without any documentation (more than 20,000 lines); 2) a GNU (a Unix-like computer operating system) development environment needs to be established and FLTK (a cross-platform GUI library made with 3D graphics programming) based GUI library

requires to be understood; and 3) COM library establishment needs to be understood. All of these tasks consume even more time than the first idea.

To bypass the complicated hardware controlling difficulties aforementioned, we consider using C# .NET to execute this console application externally in the background since the bdc-comm provides a command line console to control the motor with several simple commands. The input of the commands and the output of the real-time data can be redirected by System.Diagnostics.Process Class under .NET Framework for C#. Therefore, the motor controlling can be easily realized by calling the commands we want in the background.

On the other hand, considering the coupling with different components, the generalization of the code, and the simplicity of modification for different system layout, three design patterns are required to be applied in the programming platform of hardware testbed: namely, prototype pattern, singleton pattern, and observer pattern.

Prototype pattern is a pattern that specifies the kinds of objects to create a prototypical instance, and create new objects by copying this prototype (Wikipedia contributors 2012c). Since there are multiple machines and buffers in the system, we can develop two prototypes for one machine and one buffer, respectively. Other machines and buffers can be cloned from these prototypes. Therefore we only have to modify the machine or buffer prototypes to add parameters or functional modules.

Singleton pattern ensures that a class has only one instance, and provides a global point of access to it (Wikipedia contributors 2012d). For example, the control module for Jaguar should be a global object. The timing controller takes the responsibility of machine states decision-making and buffer-updating, which should also considered being a global object.

The observer pattern is that, an object (called the subject) maintains a list of its dependents (called observers), and notifies them automatically of any state changes, usually by calling one of their methods. It is mainly used to implement distributed event handling systems (Wikipedia contributors 2012a). Especially within the interaction between the presentation tier and the business logic tier. When the machine or buffer is changing to a new status, the GUI should be notified to change the displayed data and status automatically at the same time.

To realize these three patterns, C#, a simple, modern, general-purpose, and object-oriented programming language under the .NET Framework, which is helpful to shorten the development cycle, is selected under a WinForms-based application developed by Visual Studio 2010.

In summary, our development environment is the language of C# under .NET Framework with the Integrated Development Environment (IDE) of Visual Studio 2010.

3.4.3. Class Diagram

Based on the language and coding environment selection in Chapter 3.4.2, we introduce the concrete code structure in this section in detail. As is known to all, within Object-Oriented

Programming, each class (module) is designed as independently as it could to reduce coupling. The relationship and communications of all classes declared in the system are shown in Figure 27 and the functionality of each class is introduced as follows:

Buffer.cs

This is the prototype class of buffers. Instances are created to represent buffer objects. Build-in runtime parameters of ID, Capacity, InitialQty, and CurrentQty are declared to represent the buffer number, buffer capacity, buffer initial contents, and current contents, respectively.

BufferParameters.cs

This is a collection of static members of buffer instance, so the buffer data could be treated as global variables. A Buffer Type array is created for convenience so in future program "BufferParameters.B[i]" can be called for the traversal of all buffer parameters.

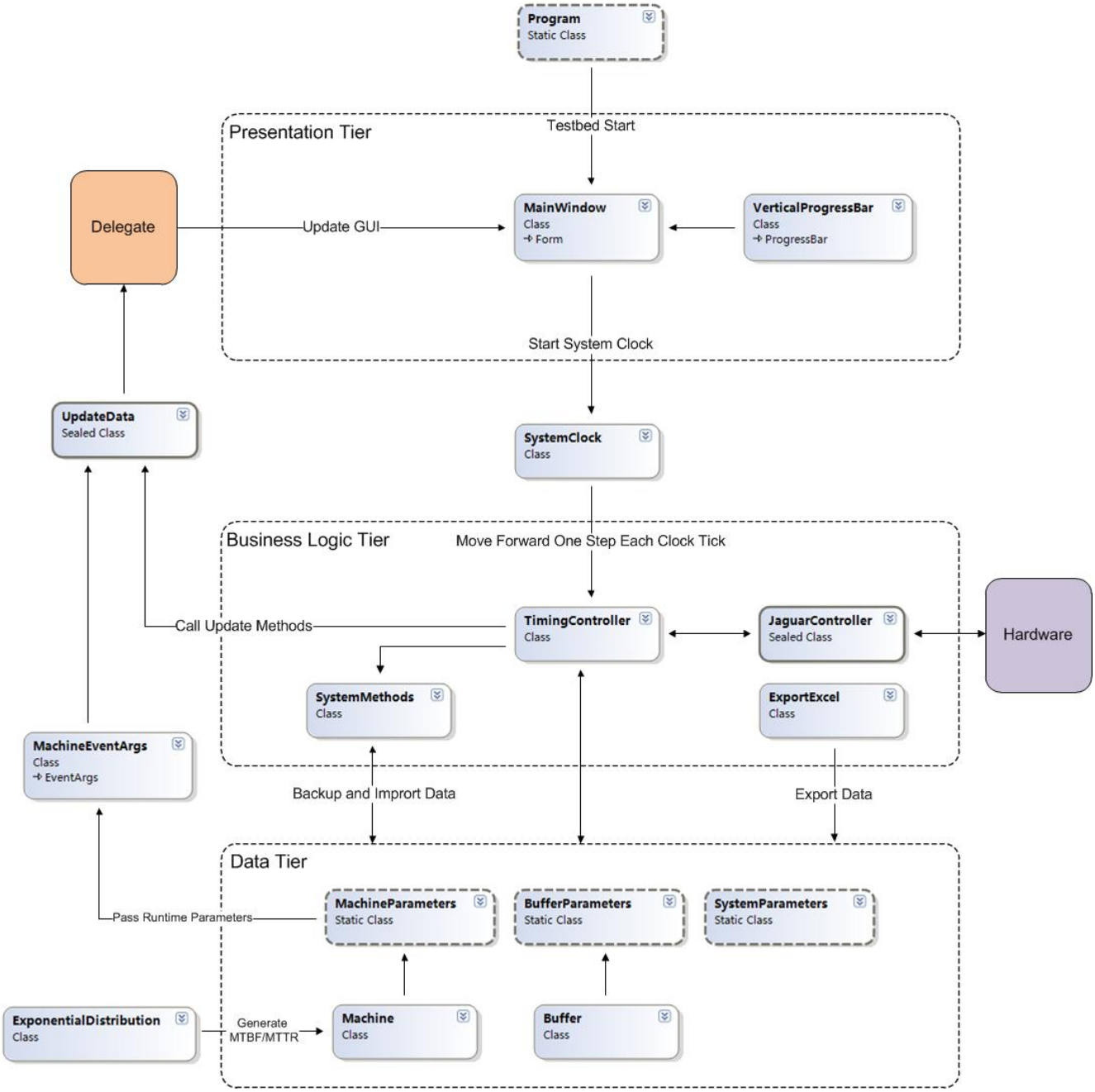


Figure 27 Class Diagram of Class Relationships and Communications

DelegateDeclaration.cs

This is a collection of four global delegate declarations. Four update delegates are declared for updating GUI in multithreading environment. UpdateGUIHandler Delegate is used to update the machine states, power levels, progress bars of power, and information list view. UpdateBufferHandler Delegate is used to update the buffer contents and vertical progress bars of buffer indictor. UpdateThroughputHandler Delegate is used to update the throughput quantity and vertical progress bar of throughput indictor. UpdateSystemClockHandler Delegate is used to update the system progress bar, as well as the system clock on the right top corner of the user interface.

ExponentialDistribution.cs

This is a class for generating exponentially distributed numbers. These numbers are translated into minutes for representing the MTBF and MTTR parameters for machines. They are generated in the constructor of Machine Class to initialize the runtime parameters.

ExportExcel.cs

This is a class used for exporting runtime data after one shift of baseline and adjusted scenario is finished. Because of the large volume of IO requests, single cell value assigning will extremely slow down the system and freeze the application. So we use a two-dimension object array for

assigning values. Notice that Excel.Application.Visible should be turned off during the operation and turned on again after the operation, otherwise the Excel is likely to be frozen.

JaguarController.cs

This is a singleton pattern class that communicates with bdc-comm.exe to execute all Jaguar control commands. Bdc-comm.exe is a command line application developed by Texas Instruments to control and monitor the Jaguar. RedirectStandardInput Method in Process Class is called to input commands to the console of bdc-comm.exe in the background. Process.BeginOutputReadLine Method begins to asynchronously read output string on the redirected StandardOutput stream of the bdc-comm.exe once the Start Method is called by the button in MainWindow Form. Therefore the voltage and current could be obtained through the input of "stat vout2", and "stat cur", respectively. ExtractValue Method is called to analyze and extract the exact voltage or current after the string obtained from the StandardOutput stream.

Machine.cs

This is the prototype class of machines. Instances can be created to represent machine objects. Build-in runtime parameters are declared and different warm up times and cycle times are customized in the set of InitializeMachine Methods. In addition, in the constructor of this class, GenerateMTBFMTTR Method is called to generate random MTBF and MTTR, which is only executed on the instantiation of baseline. Note that the MTBF ArrayList is the beginning time

point of machine failure, while the MTTR ArrayList is the end time point of machine failure. Considering the warm up times, RecoveredTime ArrayList is also generated.

Two important properties of UpstreamBuffer and DownstreamBuffer are declared in this class. They are Buffer Class instances. Once the UpstreamBuffer and DownstreamBuffer instance is assigned in the constructor, the manufacturing line relationship is determined. Take our case for example; both machine 4 and machine 5 will be assigned with the code of:

```
UpstreamBuffer = BufferParameters.B3;
DownstreamBuffer = BufferParameters.B4;
```

Therefore the parallel relationship of machine 4 and machine 5 are determined.

MachineEventArgs.cs

This is a class for carrying machine runtime event arguments. In observer pattern, event argument is an important element which can pass the necessary data to the observer. Without this class, code reusability decreases and sometimes it is difficult to pass the parameters to the event handler.

MachineParameters.cs

This is a collection of static members of machine instance, so that machine data could be visited and stored as global variables. A Machine Type array is created for convenience so in future

program "MachineParameters.M[i]" can be called for the traversal of all machine parameters. Apart from that, MBackup[i] array is also declared to backup and store the baseline machine runtime parameters including: MTBF, MTTR, beginning/ending time of blockage and starvation, as well as real-time voltage and current. A Reset Method is included to re-instantiate the same seven new machine instances after baseline is finished

MainWindow.cs

This is the main user interface to present the interaction of machine and buffer data, along with the system parameters. The design is shown in Figure 28.

Meanwhile, this class includes batches of GUI updating methods, which can be called by the update Delegates declared in DelegateDeclaration Class when the machine or buffer status has changed. Multithreaded asynchronous invoke methods can be called for updating since the working thread is different from the main UI thread. Otherwise, the cross-thread operation exception will occur.

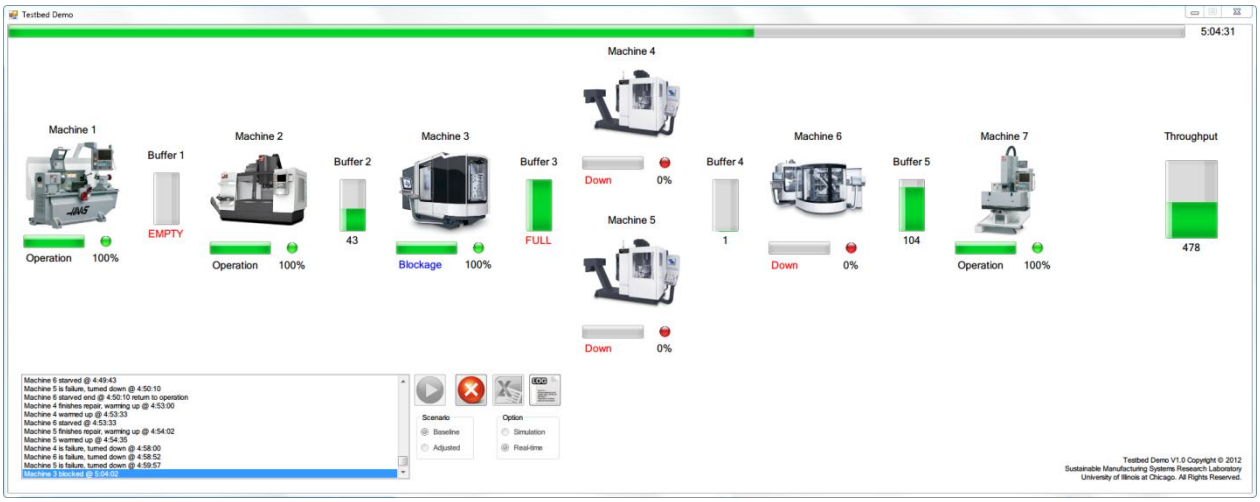


Figure 28 MainWindow Form User Interface

Program.cs

This is the main entry point for the programming platform.

SystemClock.cs

This is the global system clock generator. Each system clock tick moves forward the machine ClockTime one step (Details in Chapter 3.4.4). There are two kinds of clocks in this class. One is the real clock, with the time interval followed by SystemParameters.ClockRate. The other is the simulated clock generated by a for-loop. They are launched by Start Method and StartSim Method, respectively.

SystemMethods.cs

This is some extra system level methods class. Three methods 1) AmendMTBFBlockageStarvationEndTime, 2) BackupData, and 3) ImportRuntimeData are included. Sometimes blockage and starvation last until simulation ends thus the BlockageEnd and StarvationEnd cannot be captured so the first method is used to make up these losses. The second method is used to retrieve data from machine instances to backup machine instances, including: MTBF, MTTR, RecoveredTime, BlockageBegin, BlockageEnd, StarvationBegin, StarvationEnd after the baseline is finished. The third method imports runtime data from the backup machine instances to the current reset machine instances in adjusted scenario.

SystemParameters.cs

Parameters other than machine parameters and buffer parameters are read from and stored into this class. The most important parameter is ClockRate, used to adjust the simulation or real-time speed of the testbed. A Reset Method is also included to clear throughput to zero after the baseline is finished.

TimingController.cs

This is the software core function unit for handling runtime logic. All runtime decisions will be determined in this class (Details in Chapter 3.4.5), such as when the machine is in failure, blockage, starvation, or operation. In addition, measuring and converting to obtain the energy

consumption is also executed in this class. After every step, this class will notify MainWindow Form to update the GUI presenting.

UpdateData.cs

This is a collection of delegated updating methods for different parts of control in MainWindow Class.

VerticalProgressBar.cs

This is a customized control to visualize the buffer and throughput content. ProgressBar Control is usually horizontally oriented. Here we reconfigure the process bar into a vertical one for better visualization.

3.4.4. System Clock

The challenging obstacle of programming platform of the hardware testbed is how we could control the timing and execute system behaviors at certain time points. According to our initial attempts, we used several timers to control the timing. For example, for machine 1, we designated timer1 for warm up time, timer2 for cycle time, timer3 for random failure time, timer4 for random repair time. For the power level transition time we also designated corresponding timers. Totally more than 15 timers were required for just one machine prototype. Considering multiple machines and other system timer requirements, a great number of timers are required. Therefore, a large amount of system resources are requested which occupies

unexpected memory consumption and reduces the system response speed. In addition, since each timer needs to create a new thread, therefore thread synchronization needs to be considered to prevent data conflict. As a result, the risk of testbed crash increases and the system robustness decreases.

The solution we proposed here to circumvent the above challenge is to discretize the continuous time in real world into finite time slices. Every time slice is defined as a Simulation Step with length of 1000ms, which means the system clock ticks every 1,000ms. It can be used to represent a unit of simulation time period as shown in Figure 29. The red line corresponds with the simulation clock with the same pace of the real world time, namely, 1000ms per Simulation Step; the blue line represents 2,000ms per Simulation Step; and the green line denotes 500ms per Simulation Step respectively.

From the systematic perspective, there are two clocks in our system. One is the simulation clock and the other is the real world clock. Each machine has their individual ClockStep properties, which steps followed by the simulation clock, in other word, each machine moves their clocks one step forward simultaneously by a traversal method when the simulation clock ticks.

The total simulation time of an 8-hour shift is discretized into milliseconds:

$$8h \times 3,600s / h \times 1,000ms / s = 28,800,000ms \tag{3.2}$$

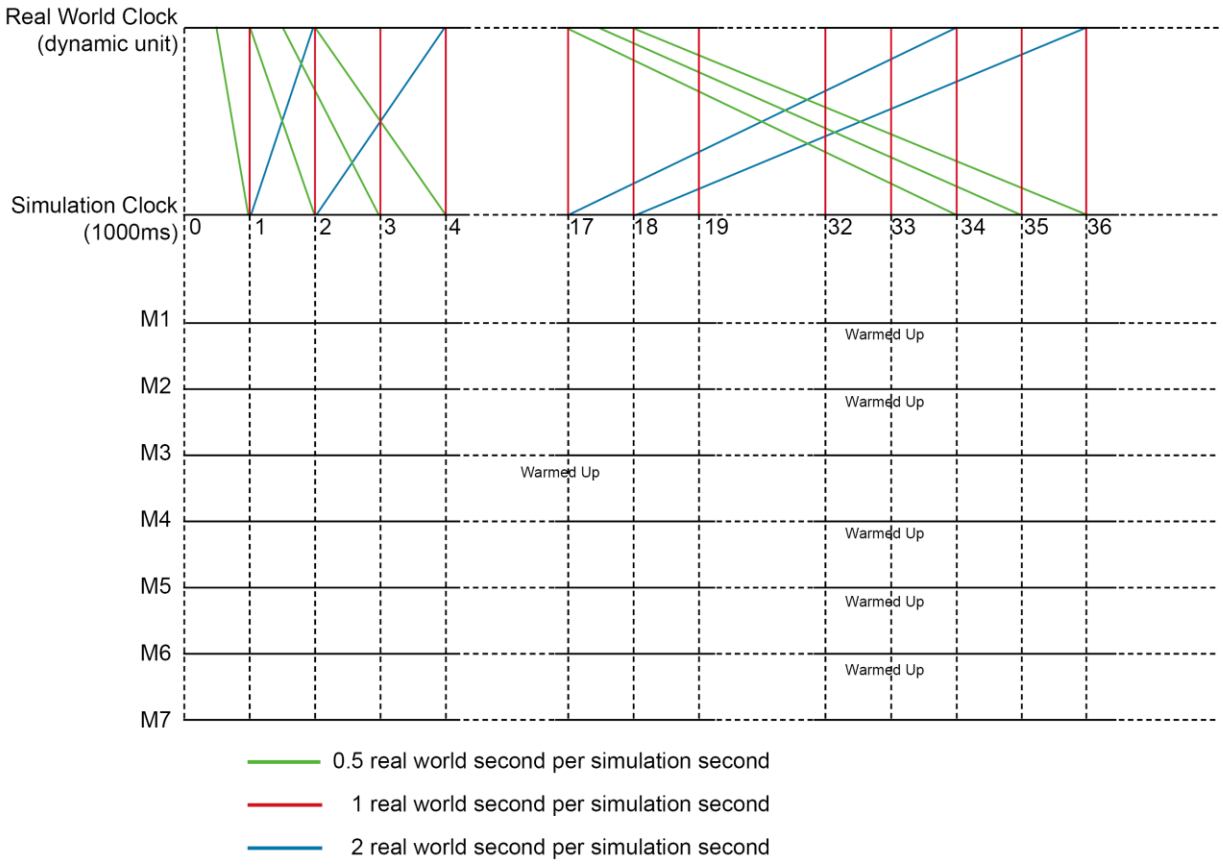


Figure 29 System Clock Compared with Real World Clock and Simulation Clock

Therefore we could discretize the system into 28,800,000 discrete steps. For example, if the system is currently running at the time point of 3 hours 45 minutes, then the ClockStep for each machine should be:

$$\left(3h\times 3{,}600s/h\times 1{,}000ms/s+45min\times 60s/h\times 1{,}000ms/s\right)\times 1step/ms=13{,}500{,}000step \quad (3.3)$$

Thus, we can use one global timer to represent the real world clock. When the Timer.Elapsed Event occurs at every 1,000ms, it traverses the ClockStep properties of all seven machine instances. For example, the warm up time of machine 3 is 17,000ms and therefore when the real world clock ticks the 17th second, the simulation clock also reaches 17,000ms. Thus, when the traversal method scans to machine 3, it detects that machine 3 has finished warm up and so the following system behaviors can be executed.

The advantages of this solution are obvious: 1) save huge system resources and memory consumptions; 2) prevent multi-timer conflict to secure the system robustness; 3) record and reoccur MTBF, MTTR, blockage, and starvation time point easily; and 4) make sure that hardware testbed speed is adjustable as software simulation testbed.

3.4.5. Runtime Logic

Within the TimingController Class, the Run Method can handle all runtime logic. Figure 30 shows the running logic flow chart for each ClockStep in Machine Class. Several critical time point designs are also demonstrated below.

The corresponding code for this flow chart is as follows:

```
public void Run()
{
     if (IsStartupWarmedUp())
```

```
{
     if (!Mach.IsStartupWarmedUp)
     {
          DoStartupWarmedUp();
     }

     if (IsSimulationEnd())
     {
          End();
     }
     else
     {
          if (IsMTBF())
          {
               DoMTBF();
          }
          else
          {
               if (IsStarvation() || IsBlockage())
               {
                    if (IsStarvation())
                    {
                         if (IsBaseline())
                         {
                              DoStarvationBaseline();
```

```
                }
                else
                {
                        DoStarvationAdjusted();
                }
        }
        if (IsBlockage())
        {
                if (IsBaseline())
                {
                        DoBlockageBaseline();
                }
                else
                {
                        DoBlockageAdjusted();
                }
        }
}
else
{
        if (IsOperationPoint())
        {
                DoOperation();
        }
}
```

```
                }
            }
        }
        else
        {
            DoWarmingUp();
        }

        Measuring();

UpdateSystemClock();
}
```

We did not write any real codes in this Run Method, but put the corresponding handling methods in it. The reason is that we wish to keep the code clean and easy to understand, and separate each module independently.

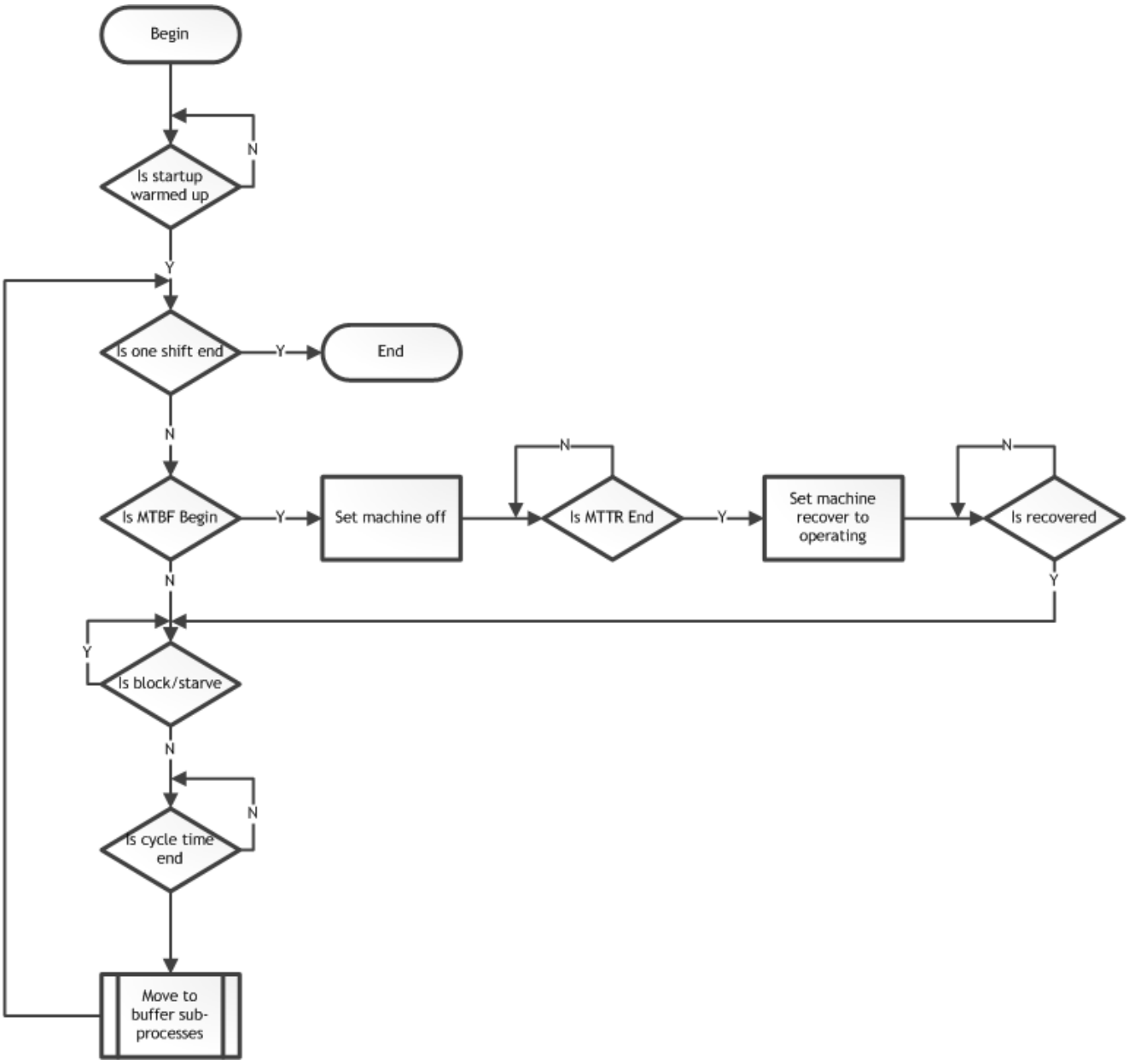


Figure 30 Runtime Logic Flow Chart

Downtime

In the constructor of Machine Class, random MTBF and MTTR time points are generated followed by an exponential distribution for each machine instance. These data are stored into two ArrayList Objects initially.

Blockage and Starvation

Since no more parts will be produced during the machine downtime, the upstream buffer keeps accumulating until it is full and the downstream buffer keeps dissipating until it is empty. And once the TimingController Class runs into a step which satisfies the blockage or starvation conditions, it prevents the next step of the certain machine from operating. At the same time, it notifies the UpdateGUI Delegate to present some information in the MainWindow Form.

The identification of blockage and starvation is a little bit complicated. Take blockage for example: the IsBlocakge Method will first check whether the DownstreamBuffer is full. If it is full, then check whether the downstream machine is breakdown or blockage. Only two of these conditions are satisfied, this machine can be flagged as blockage.

When this blockage machine recovers from blockage, as well as failure, it must double check whether the upstream machine is also blocked. If so, update the OperationPoint of the upstream machine. Otherwise, the upstream machine won’t wake up.

3.5. Case Study

To verify the effectiveness of the hardware testbed proposed in previous sections and strengthen the results of Chapter 2, a case study with same reliability data as in Chapter 2 is implemented on our hardware testbed. Energy related parameters are measured through JaguarController Class according to the hardware testbed instead of previous plant-based power parameters as input to the case (see TABLE VII).

TABLE VII
PARAMETERS FOR DIFFERENT ENERGY LEVELS IN HARDWARE TESTBED

Energy Level	Current (A)	Voltage (V)	Power (W)
10%	1.64	2.35	3.854
30%	2.86	7.05	20.163
50%	3.43	11.72	40.200
100%	2.83	23.35	66.080

TABLE VIII
BASIC SETTINGS FOR MACHINES IN HARDWARE TESTBED

	MTBF (Simulation Step)	MTTR (Simulation Step)	Cycle Time (Simulation Step)	Warm up Time (Simulation Step)
M_1	5,100,000	620,400	30,000	33,000
M_2	25,974,000	297,000	27,000	33,000
M_3	2,094,000	436,200	27,000	17,000
M_4	1,068,000	408,600	58,000	33,000
M_5	798,000	462,000	59,000	33,000
M_6	4,788,000	629,400	26,000	33,000
M_7	3,054,000	477,600	30,000	33,000

As the same machine reliability parameters given in the case study of Chapter 2.4, we converted the MTBF, MTTR, Cycle Time, and Warm up Time into the unit of simulation step, as shown in TABLE VIII. Buffer parameters are the same as the parameters in TABLE II.

The throughput is compared between baseline model and adjustment model. It can be observed from TABLE IX that no throughput loss occurs when we execute power state adjustment compared to the baseline scenario.

The energy consumption is compared between the adjustment scenario and the baseline model is described in TABLE X. It can be observed that averagely, approximate 12.6% of the total energy can be conserved by adjusting machines power level discretely when machines are blocked or starved, which is quite close to the simulation result in Chapter 2.

TABLE IX and TABLE X both compared the results between software testbed demonstrated in Chapter 2 and hardware testbed in this chapter. It can be observed that the confidence interval of both throughput and energy saving potential of hardware testbed is within the confidence interval of the results of software testbed.

TABLE IX
COMPARISON OF THROUGHPUT BETWEEN BASELINE AND POWER ADJUSTMENT MODEL IN HARDWARE TESTBED

		Throughput of Baseline Model	Throughput of Power Adjustment Model
Hardware Testbed	Throughput	633	630
	95% Confidence Interval	(616, 650)	(614, 646)
Software Testbed	Throughput	636	634
	95% Confidence Interval	（622, 650）	（618, 650）

TABLE X
COMPARISON OF ENERGY CONSUMPTION BETWEEN BASELINE AND POWER ADJUSTMENT MODEL IN HARDWARE TESTBED

	Energy Consumption of Baseline Model	Energy Consumption of Power Adjustment Model	Energy Saving Potential in Hardware Testbed	Energy Saving Potential in Software Testbed
Electricity Consumed (kJ)	11,110	9,706	12.6%	13.8%
95% Confidence Interval	(11,034, 11,186)	(9,576, 9,836)	(11.8%, 13.4%)	(11.3%, 16.3%)

3.6. Conclusion

In this chapter, a hardware testbed is developed. All the details of the design and development of the testbed including the testbed hierarchy, hardware infrastructure, and programming platform are elaborated. The design difficulties and challenging obstacles, as well as the corresponding solutions are also explained. A case study with same layout and reliability parameters as the one in Chapter 2 is implemented and a very close result is achieved. This indicates 1) the result of software testbed is convincible; and 2) the hardware testbed realizes the experimental analysis of real-time energy control potentials for sustainable manufacturing systems.

Furthermore, the design of the hardware testbed considers the flexibility. The object-oriented programming and parameterization of the programming platform secures the code reusability and simplicity of modification for testbed rearrangements. It is convenient to fit for the validation of different manufacturing layout and various cases in future research.

4. CONCLUSION AND FUTURE WORK

In this thesis, a simulation-based algorithm to investigate the energy saving potentials for sustainable manufacturing systems by implementing energy control strategy during machine idle periods is developed. The experimental framework including both software and hardware testbeds are established. The feasibility of the implementation of the proposed algorithm in real hardware environment is also verified. The results of case study on both testbeds illustrate that approximately 13% energy saving potential can be achieved for a seven-machine and five-buffer hybrid system without compromising system throughput. The consistent results obtained from both testbeds also strengthen the credibility of the saving potential and the effectiveness of both testbeds.

At the same time, the flexibility is fully considered during development stage by using object-oriented programming and parameterization of the programming platform, which secures the generalization of the codes and simplicity of the structure. As a result, the revising, adding, and removing of individual module is easy to be handled and thus the developed testbed can be simply modified to adapt to different system layout or system parameters. Therefore, the future research on different fields for manufacturing systems can be performed without major retrofitting.

For the future study, analytical methods to obtain a reliable opportunity window estimation which can be utilized for energy control and the sensitivity analysis of the contribution of energy consumption of each machine to the entire system can be focused.

REFERENCES

Dahmus, J., and T. Gutowski. 2004. “An Environmental Analysis of Machining.” In *Proceedings of IMECE04 ASME International Mechanical Engineering Congress and Exposition*. Anaheim, California.

Dietmair, Anton, and Alexander Verl. 2009. “A Generic Energy Consumption Model for Decision Making and Energy Efficiency Optimisation in Manufacturing.” *International Journal of Sustainable Engineering* 2 (2): 123–133. doi:10.1080/19397030902947041.

Draganescu, F., M. Gheorghe, and C.V. Doicin. 2003. “Models of Machine Tool Efficiency and Specific Consumed Energy.” *Journal of Materials Processing Technology* 141 (1): 9–15. doi:10.1016/S0924-0136(02)00930-5.

EIA. 2006. “Manufacturing Energy Consumption Survey (MECS).” http://205.254.135.7/emeu/mecs/contents.html.

EIA. 2011. *Annual Energy Review 2010*. Washington, DC: The U.S. Energy Information Administration (EIA). http://www.eia.gov/totalenergy/data/annual/pdf/aer.pdf.

Enerdata. 2011. *Yearbook 2011 : Global Energy Market Review in 2010*. Enerdata. http://www.enerdata.net/enerdatauk/press-and-publication/publications/world-energy-statistics-supply-and-demand.php.

Galitsky, Christina, and Ernst Worrell. 2008. “Energy Efficiency Improvement and Cost Saving Opportunities for the Vehicle Assembly Industry”. ERNEST ORLANDO LAWRENCE BERKELEY NATIONAL LABORATORY. http://www.energystar.gov/ia/business/industry/LBNL-50939.pdf.

IEA. 2008. *Worldwide Trends in Energy Use and Efficiency: Key Insights from IEA Indicator Analysis*. The International Energy Agency (IEA).

IEA. 2011. *Key World Energy Statistics 2011*. The International Energy Agency (IEA). http://www.iea.org/textbase/nppdf/free/2010/key_stats_2010.pdf.

Kanako, Tanaka. 2011. “Review of Policies and Measures for Energy Efficiency in Industry Sector.” *Energy Policy* 39 (10): 6532–6550. doi:10.1016/j.enpol.2011.07.058.

Kankana, Mukherjee. 2008. “Energy Use Efficiency in U.S. Manufacturing: A Nonparametric Analysis.” *Energy Economics* 30 (1): 76–96. doi:10.1016/j.eneco.2006.11.004.

LEESON. 2010. “DC Motor SCR Rated Low Voltage”. LEESON Electric Corporation. http://www.leeson.com/Literature/pdf/1050/DCMotorsSCRRatedLowVoltage.pdf.

Li, Jingshan, Semyon M. Meerkov, and Liang Zhang. 2010. “Production Systems Engineering: Problems, Solutions, and Applications.” *Annual Reviews in Control* 34 (1) (April): 73–88. doi:10.1016/j.arcontrol.2010.02.003.

Li, Lin, Saumil Ambani, and Jun Ni. 2009. “Plant-level Maintenance Decision Support System for Throughput Improvement.” *International Journal of Production Research* 47 (24): 7047–7061. doi:10.1080/00207540802375705.

Li, Lin, Qing Chang, Jun Ni, and Stephan Biller. 2009. “Real Time Production Improvement Through Bottleneck Control.” *International Journal of Production Research* 47 (21): 6145–6158. doi:10.1080/00207540802244240.

Li, Lin, Zeyi Sun, Haoxiang Yang, and Fangming Gu. 2012. “A Simulation Based Approach to Energy Efficiency Improvement Study of Sustainable Manufacturing Systems.” In *Proceedings of the ASME 2012 International Manufacturing Science and Engineering Conference*. Notre Dame, IN.

Lu, Lingbo, Yang Liu, Jingshan Li, C. Chang, S. Biller, and Guoxian Xiao. 2011. “A real-time maintenance scheduling policy in serial production lines.” In *2011 9th World Congress on Intelligent Control and Automation (WCICA)*, 36–41. IEEE. doi:10.1109/WCICA.2011.5970578.

Mouzon, Gilles, and Mehmet B. Yildirim. 2008. “A Framework to Minimise Total Energy Consumption and Total Tardiness on a Single Machine.” *International Journal of Sustainable Engineering* 1 (2): 105–116. doi:10.1080/19397030802257236.

Park, Cheol-Woo, Kye-Si Kwon, Wook-Bae Kim, Byung-Kwon Min, Sung-Jun Park, In-Ha Sung, Young Sik Yoon, Kyung-Soo Lee, Jong-Hang Lee, and Jongwon Seok. 2009. “Energy consumption reduction technology in manufacturing - A selective review of policies, standards, and research.” *International Journal of Precision Engineering and Manufacturing* 10 (5) (December): 151–173. doi:10.1007/s12541-009-0107-z.

Sun, Z., S. Biller, F. Gu, and L. Li. 2011. “Energy Consumption Reduction for Sustainable Manufacturing Systems Considering Machines with Multiple-power States.” In *Proceedings of the ASME 2011 International Manufacturing Science and Engineering Conference*. Corvallis, Oregon, USA.

TI. 2012a. “RDK-BDC24 README FIRST”. Texas Instruments. http://www.ti.com/lit/ml/spmu133b/spmu133b.pdf.

TI. 2012b. “MDL-BDC24 Getting Started Guide”. Texas Instruments. http://www.ti.com/tool/mdl-bdc24&DCMP=STELLARIS&.

Wikipedia contributors. 2012a. “Observer Pattern.” *Wikipedia, the Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/w/index.php?title=Observer_pattern&oldid=488920385.

Wikipedia contributors. 2012b. “Multitier Architecture.” *Wikipedia, the Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/w/index.php?title=Multitier_architecture&oldid=489350781.

Wikipedia contributors. 2012c. “Prototype Pattern.” *Wikipedia, the Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/w/index.php?title=Prototype_pattern&oldid=488319858.

Wikipedia contributors. 2012d. “Software Design Pattern.” *Wikipedia, the Free Encyclopedia*. Wikimedia Foundation, Inc. http://en.wikipedia.org/w/index.php?title=Software_design_pattern&oldid=489266791.

Worrell, Ernst, Lenny Bernstein, Joyashree Roy, Lynn Price, and Jochen Harnisch. 2009. “Industrial Energy Efficiency and Climate Change Mitigation.” *Energy Efficiency* 2 (2): 109–123. doi:10.1007/s12053-008-9032-8.

VITA

NAME:	Haoxiang Yang
EUDCATION:	B.Eng., Mechanical Engineering and Automation, Dalian Jiaotong University, Dalian, P.R.China, 2010
	M.S., Industrial Engineering, University of Illinois at Chicago, Chicago, Illinois, 2012
EXPERIENCE:	Graduate Assistant, Urban Health Program, University of Illinois at Chicago, Chicago, Illinois, 2011 - 2012
	Intern, Dalian Locomotive and Rolling Stock Co., Ltd. CNR Group, Dalian, P.R.China, 2009
	Intern, Dongfeng Chaoyang Diesel Engine Co., Ltd., Chaoyang, P.R.China, 2009
	Intern, Wuxi Turbine Blade Co., Ltd., Wuxi, P.R.China, 2009
PROJECTS:	Input-Output Time Series Model for Bispectral Index and Heart Rate of Patient during Surgery
	Corn Price Prediction using Time Series Analysis
PROFESSIONAL MEMBERSHIPS:	Student Member of American Society of Mechanical Engineers 2010 - 2011
PUBLICATIONS:	Li, Lin, Zeyi Sun, Haoxiang Yang, and Fangming Gu. 2012. "A Simulation Based Approach to Energy Efficiency Improvement Study of Sustainable Manufacturing Systems." In *Proceedings of the ASME 2012 International Manufacturing Science and Engineering Conference*. Notre Dame, IN.