Thesis work for the degree of Licentiate of Technology
Sundsvall 2012

Maintenance Consideration for Long Life Cycle Embedded System

Xiaozhou Meng

Supervisors: Dr. Benny Thörnberg
Professor Mattias O'Nils

Electronics Design Division, in the
Department of Information Technology and Media
Mid Sweden University, SE-851 70 Sundsvall, Sweden

ISSN 1652-8948
Mid Sweden University Licentiate Thesis 81

ISBN 978-91-87103-14-8

stc@miun
Sensible Things that Communicate

Mittuniversitetet
MID SWEDEN UNIVERSITY

Akademisk avhandling som med tillstånd av Mittuniversitetet i Sundsvall framläggs till offentlig granskning för avläggande av teknologie Licentiate examen i elektronik torsdag den 24 Maj 2012, klockan 10:15 i sal O102, Mittuniversitetet Sundsvall. Seminariet kommer att hållas på engelska.

Maintenance Consideration for Long Life Cycle Embedded System

Xiaozhou Meng

Electronics Design Division, in the
Department of Information Technology and Media
Mid Sweden University, SE-851 70 Sundsvall
Sweden

Telephone: +46 (0)60 148592
Printed by Kopieringen Mittuniversitetet, Sundsvall, Sweden, 2012

ABSTRACT

In this thesis, the work presented is in relation to consideration to the maintenance of a long life cycle embedded system. Various issues can present problems for maintaining a long life cycle embedded system, such as component obsolescence and IP (intellectual property) portability.

For products including automotive, avionics, military application etc., the desired life cycles for these systems are many times longer than the obsolescence cycle for the electronic components used in the systems. The maintainability is analyzed in relation to long life cycle embedded systems for different design technologies. FPGA platform solutions are proposed in order to ease the system maintenance. Different platform cases are evaluated by analyzing the essence of each case and the consequences of different risk scenarios during system maintenance. This has shown that an FPGA platform with a vendor and device independent soft IP has the highest maintainability.

A mathematic model of obsolescence management for long life cycle embedded system maintenance is presented. This model can estimate the minimum management costs for the different system architecture and this consists of two parts. The first is to generate a graph in Matlab which is in the form of state transfer diagram. A segments table is then output from Matlab for further optimization. The second part is to find the lowest cost in the state transfer diagram, which can be viewed as a transshipment problem. Linear programming is used to calculate the minimized management cost and schedule, which is solved by Lingo. A simple Controller Area Network (CAN) controller system case study is shown in order to apply this model. The model is validated by a set of synthetic and experimentally selected values. The results provided by this are a minimized management cost and an optimized management time schedule. Test experiments of the maintenance cost responding to the interest rate and unit cost are implemented. The responses from the experiments meet our expectations.

The reuse of predefined IP can shorten development times and assist the designer to meet time-to-market (TTM) requirements. System migration between devices is unavoidable, especially when it has a long life cycle expectation, so IP portability becomes an important issue for system maintenance. An M-JPEG decoder case study is presented in the thesis. The lack of any clear separation between computation and communication is shown to limit the IP's portability with respect to different communication interfaces. A methodology is proposed to ease the interface modification and interface reuse, thus to increase the portability of an IP. Technology and tool dependent firmware IP components are also shown to limit the IP portability with respect to development tools and FPGA vendors.

SAMMANFATTNING

Denna avhandling beskriver de problem som specifikt kan uppstå vid underhåll av produkter med inbyggd elektronik och som har en lång livscykel på marknaden. Att de ingående elektroniska komponenterna ej längre finns att köpa eller att (Intellectual Property) IP-komponenter ej är portabla är några av de problem som kan uppstå.

Livscykeln för elektroniska produkter inom ex. fordon, flyg eller militär industri är oftast många gånger längre än livscykeln för de komponenter som ingår. Svårighetsgraden för underhåll av inbyggd elektronik analyseras med avseende på olika designteknologier. Programmerbara grindmatriser, FPGA förslås vara en kretsteknologi som underlättar underhåll. Olika konstruktionsplattformar utvärderas utifrån ett antal riskscenarion och dess påföljande konsekvenser. Studien visar att FPGA-teknologin med fabrikat- och kretsoberoende mjuka IP-komponenter är den designteknologi som resulterar i lägst svårighetsgrad för produktunderhåll.

En matematisk modell för minimering av kostnader orsakade av åtgärder för produktunderhåll presenteras. Modellen väljer och schemalägger ett antal underhållsåtgärder under tiden för produktens hela förväntade livscykel. På så sätt kan den totala kostnaden för produktens underhåll minimeras. Utifrån parametrar så som kostnader för omkonstruktion, lagerhållning, ränta och de ingående komponenternas förväntade livscykel genereras en tillståndsgraf i Matlab. Linjärprogrammering används därefter för att välja den kombination av underhållsåtgärder som ger lägst kostnad. Vi kan visa att ett enkelt inbyggt system bestående av en mikroprocessor och en periferienhet kan analyseras med den utvecklade modellen.

Återanvändning av IP-komponenter kan korta utvecklingstider för inbyggda elektroniksystem och underlätta för företag att snabbt nå marknader med sina produkter. I de fall produkten har en lång livscykel blir det oundvikligt att någon gång behöva flytta systemet till en ny typ av krets. IP-komponenternas portabilitet blir därför en viktig parameter när svårigheter för underhåll av en produkt skall analyseras. Brist på tydlig separation mellan beskrivning av beräkning och kommunikation begränsar IP-komponenternas portabilitet med avseende på val av kommunikationsinterface. En metodologi föreslås i syfte att underlätta modifiering av IP-komponenternas kommunikationsinterface. Teknologi- och verktygsberoende beskrivningar av IP-komponenter är också begränsande för dess portabilitet med avseende på utvecklingsverktyg och kretsfabrikat.

ACKNOWLEDGEMENTS

This research work would not have been possible without the support of many people. I wish to my express deepest regards and gratitude to my supervisor Dr. Benny Thörnberg who was abundantly helpful and offered invaluable assistance, support and guidance. Deepest gratitude are also due to Prof. Mattias O'Nils and Dr. Najeem Lawal, without whose knowledge and assistance this study would not have been successful.

I am thankful to Fanny Burman and Carolina Blomberg for the administrative support.

Further, I am thankful to my friends and colleagues Abdul Waheed Malik, Khursheed Khursheed, Muhammad Imran, Naeem Ahmad, Mohammad Anzar Alam, Jinlan Gao, Xin Cheng, Omeime Esebamen, Mazhar Hussain and all the colleagues in the department for their discussions and cooperation.

I would also like to express my gratitude to Mid Sweden University (miun), Knowledge Foundation (KK) for their financial and administrative support.

I am forever indebted to my parents for their understanding, endless patience and encouragement when it was most required.

Finally, I wishes to express my love and gratitude to my wife Yue Peng; for her understanding and endless love, through the duration of my studies.

Sundsvall, March 2012

Xiaozhou Meng

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

ADC	.....	Analog to Digital Convertor
AMS	.....	Analog/Mixed-Signal
ASIC	.....	Application Specific Instruction set Processor
BRAM	……...	Block RAM
CAN	……...	Controller Area Network
COTS	.....	Commercial Off-The-Shelf
CMOS	.....	Complimentary Metal-Oxide Semiconductor
CPLD	.....	Complex Programmable Logic Device
CPU	.....	Central Processing Unit
DAC	.....	Digital to Analog Convertor
DCR	.....	Device Control Register
DMA	.....	Direct Memory Access
DRAM	.....	Dynamic RAM
DSP	.....	Digital Signal Processing
EDK	.....	Embedded Development Kit
EEPROM	.....	Electrically Erasable Programmable Read-Only Memory
EIA	.....	Electronic Industries Association
EOI	.....	End of Image
EOL	.....	End of Life
FIFO	.....	First In First Out
FPGA	.....	Field Programmable Gate Array
FSL	.....	Fast Simplex Link
HDL	.....	Hardware Description Language
HDMI	.....	High-Definition Multimedia Interface
IC	……...	Integrated Circuits
IEEE	.....	Institute of Electrical and Electronics Engineers
IP	.....	Intellectual Property
ISA	.....	Industry Standard Architecture
JPEG	.....	Joint Photographic Experts Group
JTAG	.....	Joint Test Action Group
LGPL	.....	GNU Lesser General Public License
LP	.....	Linear Programming
LTB	.....	Last Time Buy
MCU	.....	Micro-Controller Unit
MDM	.....	MicroBlaze Debug Module
MJPEG	.....	Motion Joint Photographic Experts Group
MMU	.....	Memory Management Unit
NPI	.....	Native Port Interface
OPB	.....	On-chip Peripheral Bus

PC	.....	Personal Computer
PCI	.....	Peripheral Component Interconnect
PLB	.....	Processor Local Bus
PLC	.....	Product Life Cycle
PLD	.....	Programmable Logic Device
PROM	.....	Programmable Read-Only Memory
RAM	.....	Random Access Memory
RF	.....	Radio Frequency
RGB	.....	Red Green Blue
RISC	.....	Reduced Instruction Set Computing
ROM	.....	Read Only Memory
RTL	.....	Register Transfer Level
SDRAM	.....	Synchronous Dynamic Random Access Memory
SOC	.....	System on Chip
SPI	.....	Serial Peripheral Interface
SRAM	.....	Static Random Access Memory
TFT	.....	Thin-Film Transistor
TTM	.....	Time to Market
UART	.....	Universal Asynchronous Receiver/Transmitter
USB	.....	Universal Serial Bus
VGA	.....	Video Graphics Array

LIST OF FIGURES

LIST OF TABLES

LIST OF PAPERS

This thesis is mainly based on the following four papers, herein referred to by their Roman numerals:

Paper I **Embedded System Design with Maintenance Consideration**
Xiaozhou Meng, Benny Thörnberg, Najeem Lawal
34th International Convention on Information and Communication Technology, Electronics and Microelectronics, Opatija, Croatia, May 23-27, 2011

Paper II **Soft-IP Interface Modification Methodology**
Xiaozhou Meng, Benny Thörnberg, Najeem Lawal
Proc of 2011 International Conference on Information and Electronics Engineering, Bankok, Thailand, May 28-29, 2011

Paper III **Portability analysis of an M-JPEG decoder IP from OpenCores**
Xiaozhou Meng, Benny Thörnberg, Najeem Lawal
Proc. of 6th IEEE International Symposium on Industrial Embedded Systems, Vasteras, Sweden, Jun 15-17, 2011

Paper IV **Component Obsolescence Management Model for Long Life Cycle Embedded System**
Xiaozhou Meng, Benny Thörnberg, Leif Olsson
Submitted to IEEE Autotestcon 2012, Anheim, California, Sep 10-13, 2012

Related papers not included into this thesis:

Real-time Machine Vision System Using FPGA and Soft-core Processor
Abdul Waheed Malik, Benny Thörnberg, Xiaozhou Meng, Muhammad Imran
Accepted for publication in proceeding of Real-Time Image and Video Processing conference, SPIE Photonics Europe, April 15-19, 2012

1 INTRODUCTION

1.1 EMBEDDED SYSTEM

The embedding of microprocessors into equipment and consumer appliances started before the appearance of the PC (Personal Computer) and this process consumes the majority of microprocessors that are made today. In this way, embedded microprocessors are more deeply ingrained into everyday life than any other electronic circuit. For instance, in a well-equipped car, nearly every aspect has some form of electronic control associated with it and thus there is the need for a microprocessor within an embedded system. [1]

1.1.1 Embedded system overview

An embedded system is a kind of computer system with either one or a few dedicated specific functions. It is often embedded as part of a complete device including hardware and mechanical parts.

- An embedded system is controlled by one or more main processing cores such as microprocessor or digital signal processors (DSP), described in Figure 1-1. Microprocessors come in many different levels of sophistication and are usually classified by their word size [2].
  - An 8-bit microprocessor is designed for low-cost applications.
  - A 16-bit microprocessor is often used for more sophisticated applications that may require either longer word lengths or off-chip I/O and memory.
  - A 32-bit RISC (Reduced Instruction Set Computing) microprocessor offers very high performance for computation-intensive applications.
- Embedded systems usually contain a memory chip. The memory can be either on-chip or off-chip. [3]
  - Internal RAM (Random Access Memory) in a microprocessor is for register.
  - Internal ROM (Read Only Memory) is for program
  - External RAM is for the temporary data and stack
  - Internal caches (in some microprocessor)
  - EEPROM (Electrically Erasable Programmable Read-Only Memory) or flash memory is for data saving
  - External ROM or PROM (Programmable Read-Only Memory) is for software
  - RAM memory buffers at the interface ports
- Embedded system can communicate via peripherals, e.g. RS-232, Ethernet, CAN (Controller Area Network) - bus.

- Radio transceiver can be integrated into the system. E.g. RF (Radio Frequency) transmitter, Bluetooth transmitter.
- Analogue ADC (Analog to Digital Convertor) converts an analogue signal from a sensor to a digital signal for data processing by the embedded system. Also, DAC (Digital to Analog Convertor) can convert a digital signal to an analogue signal for the actuator.
- An embedded system usually contains one or more debug ports for system debugging.
- Human interface such as the TFT (Thin-Film Transistor) monitor, in which the keyboard is the communication channel between a human and the embedded system. E.g. TFT monitor, keyboard.
- Others hardware unit such as timers, interrupt handler etc. and are not shown in the figure.

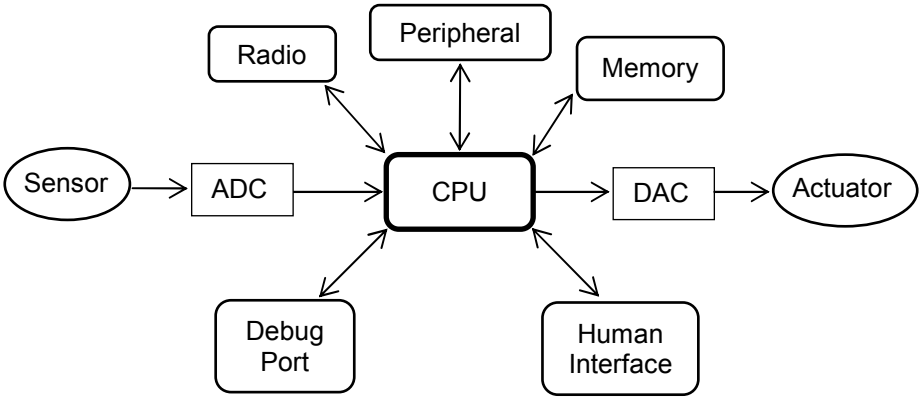

Figure 1-1. Block diagram of embedded system

A microcontroller (MCU) is a single chip containing a microprocessor, memory, timers, interrupt controller, and peripheral for different requirement etc. MCU is used in embedded system for automatic control or communication.

Software is also an important part for embedded system, however in this thesis we will focus on hardware part.

Embedded systems are widely used in commercial electronic, industry systems, avionics etc. They can be divided into several application types:

General purpose system:

- Functions are similar to those for a PC but in an embedded package.
- Video game console, set-top boxes, tablet

Control system:

- Real-time system
- Vehicle engines control system, flight control system

Signal processing system:

- Large data stream and significant computation
- Video decoder, radar

Communication system:

- Information switching and transmission
- Telephone system, network router

1.1.2 Design goals for an embedded system

A system design process has several important goals:

- Function: Mobile phone, Vehicle control etc.
- Performance: Clock frequency, response time etc.
- Manufacturing cost: Important for a consumer product to have a low retail price.
- Power consumption: Especially important for handheld devices.
- TTM: The profitable market life is time limited.
- Design cost: Development environment and engineering costs.
- Quality: Reliability, usability etc.
- Others: Maintainability, security.

1.2 EMBEDDED SYSTEM HARDWARE PLATFORM

The hardware architecture of the embedded system can be formed as a bus based system, see Fig.1-2. There could be different hardware technology platforms for an embedded system, including Application-Specific Integrated Circuit (ASIC), Field Programmable Gate Array (FPGA).

1.2.1 Bus based embedded system

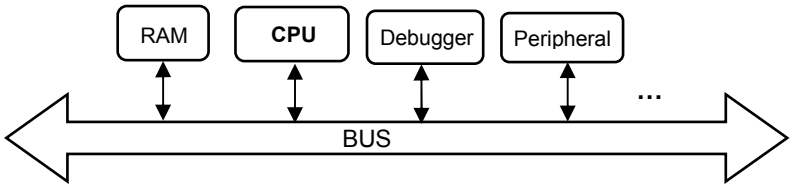

Figure 1-2. System architecture of bus based FPGA system

In this thesis, the embedded system hardware architecture is considered as a bus based system. The bus is the mechanism by which the CPU communicates with others devices in the system [2]. The bus forms the backbone of the hardware system. One of the major roles of the bus is to provide an interface in relation to the memory and other devices. Each component requires interface protocol logic to connect to the bus.

1.2.2 COTS IC hardware platforms

COTS is short for Commercial Off-The-Shelf, which means products that are ready-made and available for sale to the general public. A modern embedded system designer would prefer to use COTS IC to implement embedded system, such as MCU, DDR-SDRAM etc. These will provide the best performance and lowest power consumption since their functions and performance are analyzed before their fabrication.

1.2.3 Programmable hardware platforms

A programmable logic device or PLD is an electronic component used to build reconfigurable digital circuits. A PLD has an undefined function at the time of manufacture. The function can be defined and programmed by the user (designer) such as FPGA or CPLD (Complex Programmable Logic Device). There are several basic process technology types for FPGA including Flash , Anitfuse , EEPROM, SRAM. An antifuse based FPGA is one-time programmable, while the others are re-programmable. In this thesis, the FPGAs mentioned are those of the re-programmable type.

1.3 DEVELOPMENT ENVIRONMENT

A typical embedded system design environment is shown in Fig.1-3. The software and hardware (e.g. FPGA) development is implemented in a PC, known as a host system which is illustrated in Fig.1-3. The hardware on which the code will finally run is known as the target system. The target system connects to the host system via a UART (Universal Asynchronous Receiver/Transmitter), debugger or Ethernet etc.

The main tasks of the host system include:

- Program hardware for the target (FPGA)
- Load software programs into the target
- Start and stop program execution on the target
- Examine memory and registers on the target
- Receive debugging information from the target

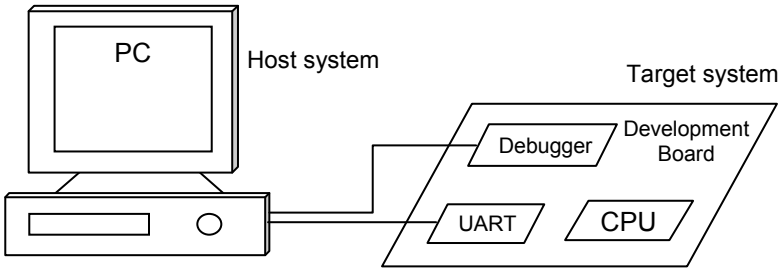

Figure 1-3. Typical embedded system design environment

1.4 PROBLEM DESCRIPTION AND MOTIVATION

Some products are not as capable of adjusting to leading-edge technology as others and to catching the development pace of consumer products. For products such as automotive, avionics, military application etc., the desired life cycle for these systems is significantly longer than the life cycle for the electronic components used in the systems. The life cycle in this thesis refers to the product field life. It is the time period for which the product is available on the market. Component obsolescence problems occur in all systems with a life cycle longer than that of one or more of their components. No publication yet offers an obsolescence management solution in relation to controlling the system maintenance costs.

The reuse of predefined Intellectual Property (IP) can lead to great success in system design and can assist the designer in meeting the TTM requirements. It is true that using IPs based FPGA device can mitigate the component obsolescence problem. However, the designer will still encounter problems if this is not used correctly. The lack of any clear separation between computation and communication will limit the IP's portability with respect to different communication interfaces. Technology and tool dependent firmware IP will limit the IP's portability with respect to development tools and FPGA vendors.

In addition to obsolescence, a system will require re-engineering if its requirements change over time or if it becomes necessary to change the specifications. The availability of newer and better architectures (processors, interconnections and interface blocks) can provide the motivation for a re-engineering of a product. System migration between devices during the system life cycle is unavoidable.

Based on these considerations, it is important that the IP has a high portability in order to maintain a long life cycle embedded system.

In this thesis, the maintenance issues including component obsolescence and IP portability etc. are discussed. Suggestions and a mathematic model are proposed in order to ease the maintenance problem for a long life cycle embedded system.

1.5 MAIN CONTRIBUTIONS

In this thesis, maintenance issues regarding to component obsolescence and IP portability are analyzed. A mathematic model for component obsolescence management is proposed.

I. Maintenance issues for an embedded system are presented. An analysis is conducted in relation to the maintainability of long life cycle embedded systems for different design technologies. The result shows that an FPGA platform with a vendor and device independent soft IP provides the highest maintainability.

II. A soft IP interface modification methodology for systems on FPGA is suggested. The methodology will ease the interface modification and interface reuse for an FPGA soft IP.

III. Maintenance issues associated with the IPs portability for the embedded FPGA system are highlighted. The lack of any clear separation between computation and communication is shown to limit the IP's portability with respect to different communication interfaces. Technology and tool dependent firmware specifications within a soft IP are also shown to limit the IP portability with respect to development tools and FPGA vendors.

IV. A mathematic model for a life cycle analysis of the long life cycle embedded system maintenance is proposed. This model is able to estimate the minimized maintenance cost caused by component obsolescence for different system architectures. An optimized maintenance schedule will also be provided by the model. It can offer maintenance strategy guidance to those designers who encounter a components obsolescence problem.

1.6 THESIS OUTLINE

Chapter 1 provides the introduction, chapter 2 addresses the maintenance issues for the embedded system, chapter 3 focuses on the maintainability analysis of the CAN controller system, chapter 4 presents a portability analysis of an FPGA IP, chapter 5 describes a component obsolescence management model for the long life cycle embedded system, chapter 6 concludes the thesis summary while chapter

7 presents a publications summary. Papers which are basis for this research work are listed at the conclusion part.

2 MAINTAINACE ISSUES FOR EMBEDDED SYSTEM

Modern embedded system designers consider various metrics during the design process, including performance, cost, power etc. However, one issue is often missing from this list, namely maintainability. Due to the rapid development in electronic technology, obsolescence and upgrade are inevitable for the majority of embedded systems which may create a variety of problems when maintaining a long life cycle system.

2.1 ELECTRONIC COMPONENT LIFE CYCLE CONCEPTS

The electronics industry is one of the fastest growing sectors of the world economy. Those new electronic components with faster speed, smaller size and lower power consumption will quickly dominate the market. Therefore, the occasion might arise in which electronic components which are the component parts of a product have a shorter life cycle than the actual life cycle of the product.

Paper [4] described the product life cycle for the following curve:

Gaussian distributions have been used by the Electronic Industries Association (EIA) as their standardized product life cycle (PLC) curve [5]. The equation of the life cycle curve is

$$f(x) = pe^{(-(x-\mu)^2/2\sigma^2)} \qquad (1.1)$$

where $f(x)$ gives values for the sales revenue of the device/technology group (or number of units shipped, or the percentage market demand), x is the year, and $f(x)$ is defined by the mean μ, which denotes the point in time of the sales-peak of the curve and the standard deviation σ. The factor p is the sales peak, the number of units shipped, or the percentage demand.

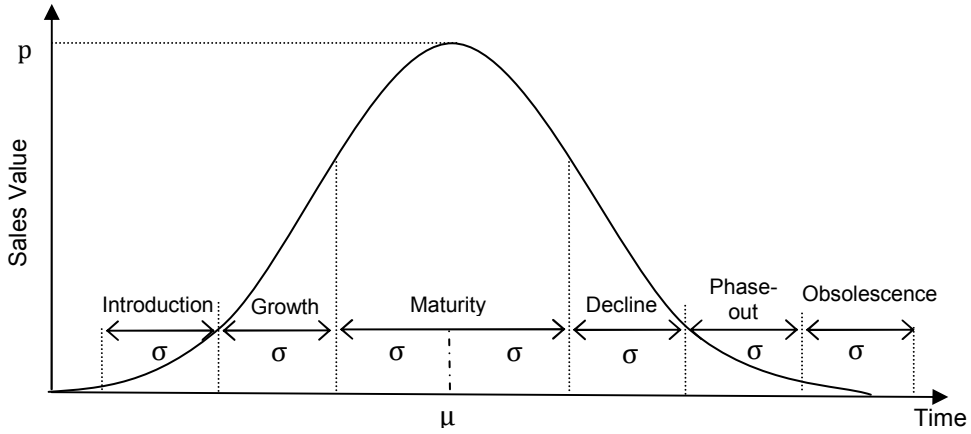

Figure 2-1. Standardized life cycle curve for a device/technology group

An electronic component life cycle can be divided into several stages: introduction, growth, maturity, decline, phase-out and obsolescence, as shown in Fig.2-1. A more in-depth explanation can be found in [6].

A. Introduction Stage

The introduction stage is the first stage of a product life cycle. The production costs are usually high because of the initial incurred design costs and low yield, frequent modifications and low or unpredictable production volumes.

B. Growth Stage

In the growth stage, the product is accepted by the market. The volume of sales increases gradually which brings about a price reduction.

C. Maturity Stage

The maturity stage is usually characterized by high-volume sales. Competitors with lower production cost may enter the market and thus, at this stage, the product will have the lowest costs throughout the entire life cycle.

D. Decline Stage

The decline stage indicates both decreasing demand and profit. During the decline stage, only a few specialized manufacturers remain in the market.

E. Phase-out Stage

During phase-out stage, a manufacturer may set a date for which the production of the part will cease. Usually, the manufacturer issues a discontinuance notice to customers, provides a last-time buy date, and suggests alternative parts or aftermarket manufacturers.

F. Discontinuance and Obsolescence

Discontinuance occurs when the manufacturer ceases to produce the components. The components may still be available in the market if the production line or stocks were bought by an aftermarket source. Obsolescence occurs at a technology level, while discontinuance occurs for a part number or manufacturer specific level.

There are some commercial databases containing component lifecycle forecast data, such as CAPS Expert from PartMiner [7]. A data mining based algorithm [8] is also proposed to improve their predictive capabilities.

2.2 OBSOLESCENCE PROBLEM FOR EMBEDDED SYSTEM

Obsolescence or end of life (EOL) is the final state of a product's life cycle when a vendor will no longer produce, sell and sustain it (i.e. no longer provided by the vendor). The growing use of COTS components and equipment increases the risk of obsolescence. The reasons for obsolescence could be technological, market, planned or environmental etc.

If a product is not popular in the market and becomes unprofitable, the manufacturer then has to commit the facilities and equipment to producing another product that results in greater profits. According to Moore's law, the number of transistors on a chip doubles every 18 to 24 months: poor planning with regards to parts obsolescence causes companies to spend progressively more in order to deal with the effects of aging systems [8]. Intel is relatively famous for its rapidly developing technology and the result of this is the rather rapid obsolescence of their products. It demonstrates a unique capability for engineering major product improvements and releasing these products into the market every 18 to 24 months. The new product will rapidly dominate the market, based on its increased performance, while still maintaining a similar price to that of its predecessor. It is possible to divide obsolescence into several types for embedded systems:

- *Peripheral interface obsolete:* The peripheral interface standard is developing. A new standard will rapidly enter the mainstream based on its improved specifications. For instance, the USB (Universal Serial Bus) has become the most popular peripheral interface standard for consumer products during the past few years. The earlier IEEE 1284 parallel interface is no longer able to be supported in most devices. Thus, it becomes a possibility that long life cycle systems will suffer from the problem of interface mismatch because of these modern peripherals.

- *Communication bus obsolete:* In this case, the communication bus is considered as an on-board or on-chip bus, which is the link between each component in the system. For example, it is not possible to support the previous Industry Standard Architecture (ISA) bus which is replaced by the Peripheral Component Interconnect (PCI) bus. For a hardware component, backward compatibility is not always guaranteed unless it incorporates an additional hardware bridge between two buses. A System-on-chip (SOC) design can also suffer from the obsolescence associated with the communication bus. For example, the On-chip Peripheral Bus (OPB) has been replaced by the Processor Local Bus (PLB) [10] for the Xilinx FPGA on-chip bus.

- *Component obsolete*: A component is a product provided by the vendor, which for the majority, contains some unique properties and cannot be replaced by a product from other vendors. Component obsolescence is a severe case since it frequently occurs. Industry experts have estimated that over 200 000 components from over 100 manufacturers became obsolete in the year 2000 [11].

- *Others:* Obsolescence issues such as obsolete development tools and test systems etc. must all be faced by designers.

In the commercial markets, electronics components in consumer electronics such as PCs or portable applications, for instance, are updated very rapidly, while in automotive, avionics, military application etc., the desired life cycle for these systems is many times longer than the obsolescence cycle for the electronic components used in the systems. For avionics and defense applications, systems face obsolescence even before they enter into service (due to the long design, manufacturing and test cycles).

It is often the case that only a part of the system is actually obsolete or requiring modification. Unfortunately, the replacement or modification is usually as difficult as designing an entire system because the system has been developed as a single entity, with much interdependence between its hardware and software [12].

Obsolete technology impacts on a company in many ways. It impacts their costs in conducting their business and hence their profits, as well as their day-to-day operations. Management must be aware of the impact of technology obsolescence on all aspects of their business, and factor this into their decision processes [13].

2.3 OTHERS MAINTENANCE ISSUES

The availability of newer and better architectures (processors, interconnections and interface blocks) can provide the motivation for the re-engineering of a product. In the commercial arena, manufacturers must re-engineer their products in order to provide those new features required by their customers, to incorporate newer technologies and standards, or to reduce costs and increase value [14].

Lower prices or better circuit technology could offer the opportunity for designers to replace the legacy components or even an entire system, so that they can reduce costs and increase value. However, it has not proved to be easy to enable either migration or replacement to occur within the different technologies as this will involve costly hardware and software redesigns.

It is a self evident truth that the customer always wants more. Manufacturers must re-engineer their products in order to provide new features required by the customers, to incorporate newer technologies and standards. Exciting new technologies can result in a better form and fit for a specification. Functions will require to be changed as will the bugs contained within the system and these require that the legacy system has to be re-engineered [14]. This is always both costly and time consuming.

2.4 CONCLUSION

This chapter has introduced electronic component life cycle concepts. Various problems in relation to maintaining a long life cycle embedded system are discussed, including obsolescence, function change requirement or technology migration etc. A more detailed analysis and a case study will be presented in subsequent chapters.

3 MAINTAINABILITY ANALYSIS OF CAN CONTROLLER SYSTEMS

CAN [15] is an industrial bus standard designed to allow microcontrollers and devices to communicate with each other. In this section, four types of CAN controller systems architecture are described. The maintainability in relation to these experimental systems was to be analyzed.

3.1 CAN BUS

The CAN bus was designed for automotive electronics and was first used in production cars in 1991. CAN is very widely used in vehicle and other industry applications. CAN runs at rate of 1Mb/s over a twisted pair connection of 40m. The bus protocol supports multiple masters on the bus.

The devices that are connected by a CAN network are typically sensors, actuators, and other control devices. These devices are not connected directly to the bus, but through a host processor and a CAN controller. Each CAN node requires a host processor, a CAN controller and a transceiver.

3.2 DESIGN CASES

The project started from a CAN controller system used in industrial construction machinery. In such a case, cost, performance and power consumption are not critical issues. However, this type of long life cycle system requires greater consideration in relation to maintainability.

The microprocessor (or MCU) and the CAN controller are two key components for this system. A peripheral of RS-232 is used for sensor reading.

Two major design platform methods are mentioned, namely the COTS IC platform and the FPGA platform. The FPGA system mentioned in this chapter is an IP based design system. A wide range of choices exists for the COTS microcontrollers and CAN controllers IC from different vendors within the marketplace.

However, for the FPGA platform, the soft microprocessors are divided into two categories:

- **Vendor dependent soft microprocessors:** Such types of soft microprocessors are usually provided by the FPGA vendors, so it is not possible to implement them on any other vendor's devices. E.g. Xilinx MicroBlaze and Altera Nios II.
- **Vendor independent soft microprocessors:** Unlike the vendor dependent soft microprocessors, these have no restrictions and can be implemented on any vendors' devices. E.g. ARM Cortex-M1 and OpenRISC from OpenCores.

3.2.1 Case 1: COTS IC based CAN controller system

For the traditional COTS IC CAN controller system, an MCU was implemented onto the board. The UART controller and the physical interface circuit RS-232 were integrated as one chip for the peripheral interface. A CAN controller and its physical interface circuit CAN transceiver were implemented on board for the CAN bus protocol. The system board architecture with its relevant components can be seen in Fig. 3-1.

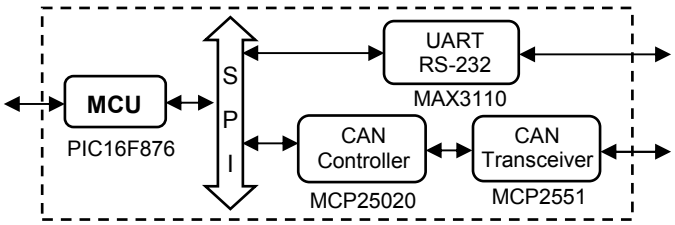

Figure 3-1. Block diagram of COTS based CAN controller system

3.2.2 Case 2: vendor specific FPGA system

A vendor specific design case based on the Xilinx Spartan3E FPGA is shown in Fig. 3-2. The MicroBlaze [16] is a soft processor core designed for Xilinx FPGAs, which can be implemented and configured by Xilinx EDK (Embedded Development Kit), as shown in Fig. 3-3. In the system, a MicroBlaze soft processor controls the UART lite and XPS-CAN via the PLB. The Xilinx 128-bit PLB v4.6 provides the bus infrastructure for connecting an optional number of PLB masters and slaves into an overall PLB system. It consists of a bus control unit, a watchdog timer, separate address, write, and read data path units, as well as an optional DCR (Device Control Register) slave interface to provide access to its

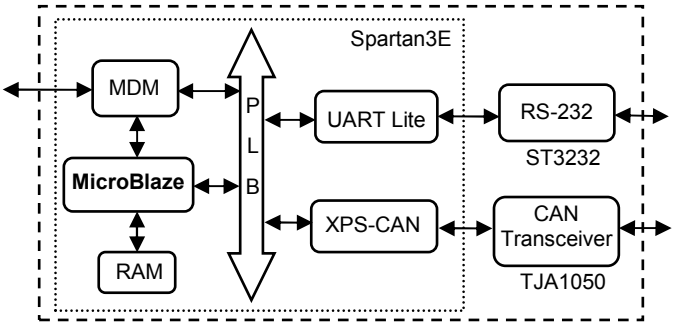

Figure 3-2. System architecture of Xilinx specific FPGA system

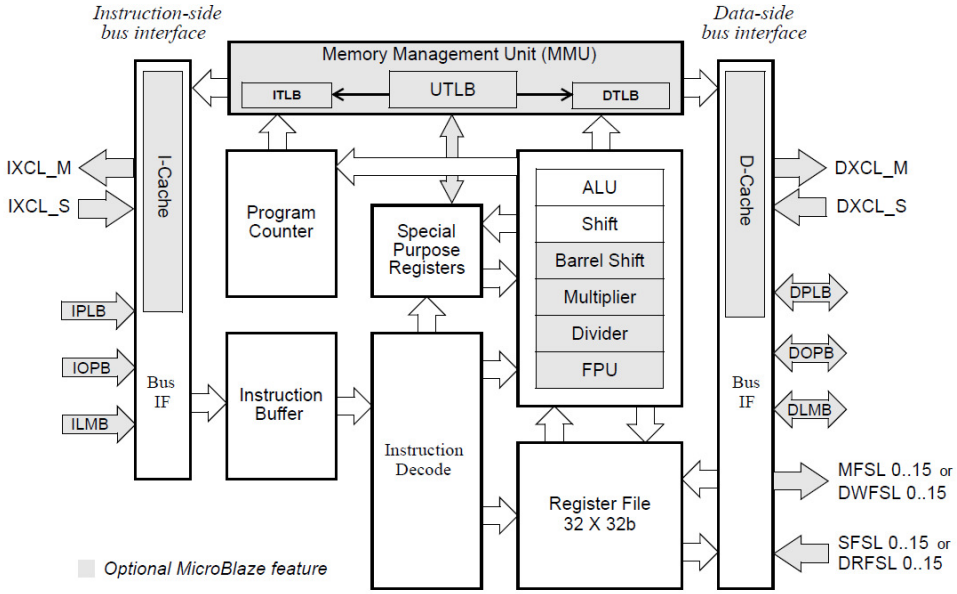

Figure 3-3. Architecture of Microblaze soft microprocessor

bus error status registers [17].

The MicroBlaze Debug Module (MDM) is used for system debugging. The development tools, FPGA devices and IPs are all provided by Xilinx. It is a relatively simple task to perform system development since the majority of the IPs are verified and can be plug-and-play. The whole design and verification process can be executed with a Xilinx tool set. The RS-232 (ST3232) and CAN transceiver (TJA1050) physical interface circuits are also integrated on board.

3.2.3 Case 3: vendor and device independent FPGA system

A vendor and device independent system is a "soft" system which can be implemented on any FPGA device. The IP could be open-source licensed or provided by third party IP providers.

Such a case is based on OpenCores soft IPs. The OpenRISC 1200 (OR1200) is a 32-bit scalar RISC with Harvard micro-architecture, 5 stage integer pipeline, virtual memory support (MMU) and basic DSP capabilities. OR1200 is licensed under a GNU Lesser General Public License (LGPL). The processor has already been verified as running on many vendors' devices and can be downloaded free and can be modified by any individual. Its architecture is shown in Fig. 3-4. The soft microprocessor is described using the Verilog HDL (Hardware Description Language). As an open source core, the design is fully public and can be downloaded and modified by any individual.

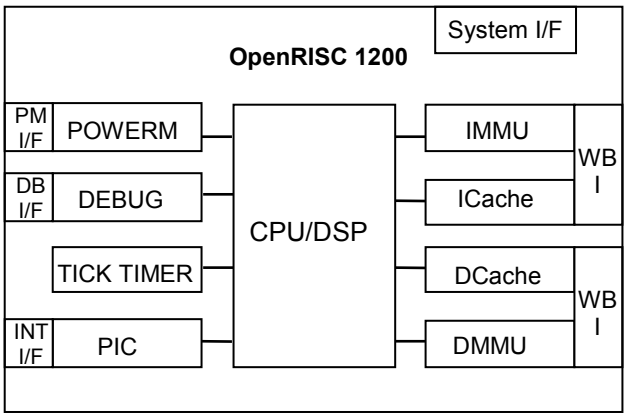

Figure 3-4. Architecture of OpenRISC 1200

The CAN controller system based on the OpenRISC 1200 is shown in Fig. 3-5. The board structure is the same as in case 2, while the FPGA on-chip architecture is different.

Every IP in the system is open source licensed in addition to being vendor and device independent. They communicate with each other via a wishbone bus, which is an open source hardware computer bus intended to allow communication between the parts of an integrated circuit communicate with each other. The aim is to allow the connection of differing cores to each other inside a chip. The Wishbone Bus is used by many designs in the OpenCores project. There are two types of Wishbone interconnects, namely the shared bus and crossbar switch. A shared bus interconnect only allows one master to communicate with one slave at the same time, while a crossbar switch may allow N masters to connect to N slaves at the same time, according to the number of implemented buses. In this particular implementation, shared bus architecture is used, which has similar features to those of a PLB bus. A debugger is used for debugging and software downloading. Different JTAG (Joint Test Action Group) cables can be used on different vendor devices and the entire system can be synthesized by using any synthesis tool. The GNU toolchain [18] which is running on a PC, including a compiler, simulator, debugger etc, is used to support the C software development as well as system debugging.

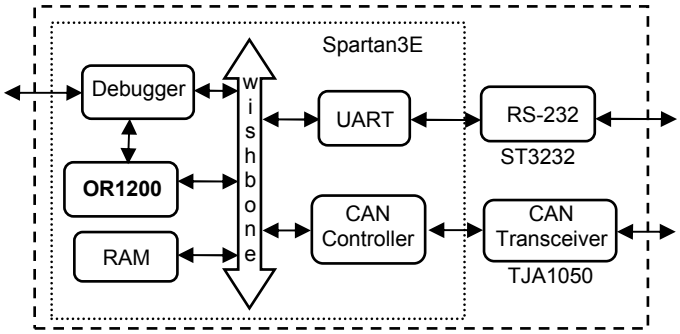

Figure 3-5. System architecture of OpenCores based FPGA system

3.2.4 Case 4: mixed FPGA system

This solution is comprised of a mixture of vendors specifics and an OpenCores platform. The board structure is the same as in case 2 and is shown in Fig. 3-6. A bridge IP is incorporated between the Xilinx PLB and the OpenCores Wishbone bus. The open-source licensed soft IPs, such as the CAN controller and the UART, can then be integrated into the system as a peripheral core for the Xilinx system. The software application is running on Microblaze microprocessor. All of the design and verification processes can be conducted in a Xilinx development environment.

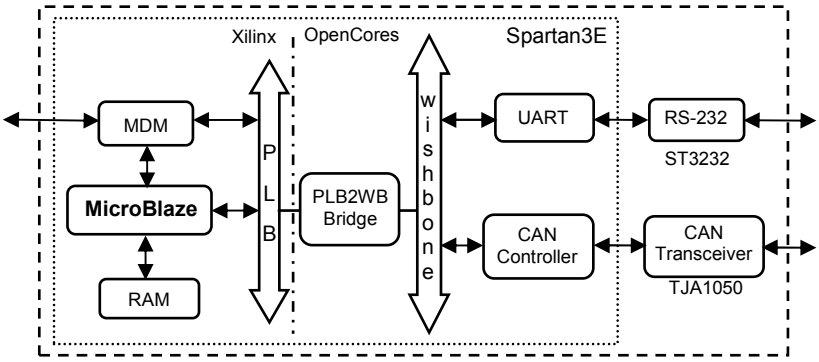

Figure 3-6. System architecture of Xilinx and OpenCores mixed FPGA system

3.3 PROTOTYPE OF FPGA BASED CAN CONTROLLER SYSTEM

A simple FPGA-CAN system demonstration is implemented. It works as a CAN node with a temperature sensor. The microprocessor, CAN controller and UART controller IP are implemented on the FPGA. The experimental system setup is described as follow (also marked in Fig. 3-7):

1. Digilent Nexys2 [19] board with Xilinx Spartan-3E FPGA.

2. The CAN transceiver (TJA1050) in the black box is the interface circuit between the CAN protocol controller and the physical bus. It connects to FPGA via Pmod ports.
3. The temperature sensor transmits temperature values through an RS-232 port on the FPGA board.
4. Xilinx platform cable USB is used for downloading and debugging.
5. USB-CAN is the interface controller between the CAN bus and the USB. The data then can be transmitted or received by the PC via a USB port.

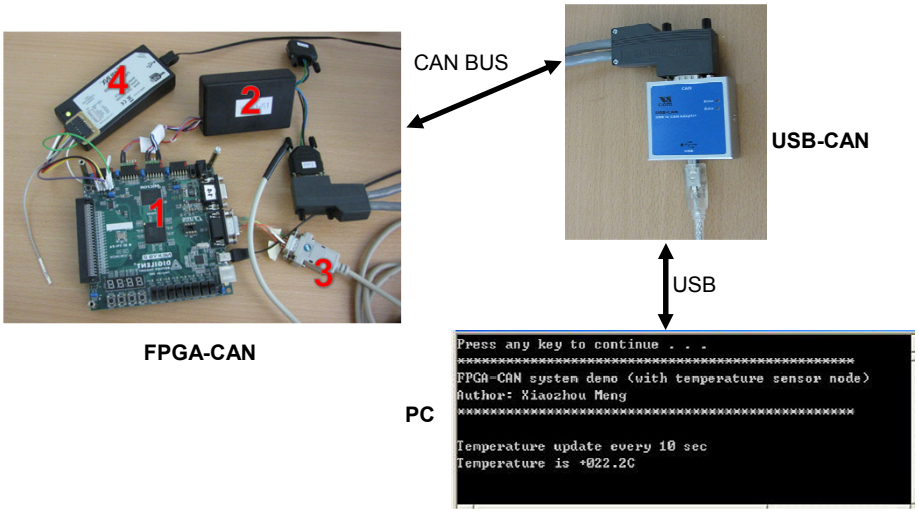

Figure 3-7. FPGA based CAN controller system prototype

The UART controller (OpenCores) can receive (interrupt based) the temperature data from the temperature sensor via the RS-232. The temperature data is then transmitted to the CAN controller (OpenCores). The CAN controller transmits the data to the CAN bus via a CAN transceiver (TJA1050). The temperature data is received by the USB-CAN and is displayed on the screen by the software running on the PC.

3.4 RISK ANALYSIS

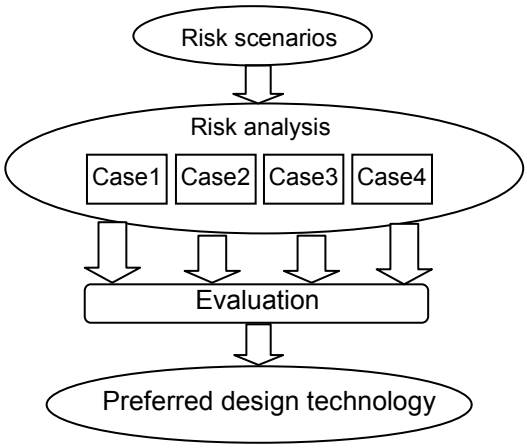

Figure 3-8. Risk analysis for different cases

A risk analysis is taken for all the cases described in section 3.1. A number of potential risk scenarios are identified. No probability is attached to the occurrence of each risk, but the consequences for the maintenance work are classified and evaluated, as shown in Fig. 3-8.

3.4.1 Risk scenarios

For the different platform cases presented in section 3.2, the system maintainability is evaluated by analyzing several potential risk scenarios. These scenarios have been developed according to the general problem issues discussed in section 2.2.

- *Microprocessor obsolescence:* Microprocessor is the heart of an embedded system. If it becomes obsolete, then there could be serious consequences.
- *Peripheral interface obsolescence:* A peripheral interface standard can be obsolete. The RS-232 interface in the system has the risk of obsolescence, including a UART controller chip, the physical interface circuit and connector.
- *Communication bus obsolescence:* The communication bus has a connection with every component in the system. Its obsolescence will soon lead to the obsolescence of all associated components.
- *Better circuit technology migration*: Better performance, lower price or being friendlier towards the environment would force the system to migrate to a new circuit technology.
- *FPGA vendor device migration:* Vendor portability is a special issue for the FPGA system. For example, if the FPGA vendor stops providing the devices (obsolescence), in this situation, the system is forced to migrate from one vendor device to another.

- *Function change requirement:* For different requirements, add, delete or modification of functions are inevitable. For example, some systems might require an Ethernet interface for data transportation.

3.4.2 Consequences

Some of the consequences with regards to the risk scenarios are classified:

- *Major board redesign:* This work is to redesign the board, including replacing or adding components, modifying the on-board bus system etc., which is costly and time consuming and can almost be equivalent to designing a new board.
- *Minor board redesign:* The redesign of a minor part of the board, including replacing or adding physical interface circuits, redefining the pins for chips, changing the connector etc., which require significantly lower design efforts as compared to those associated with a major board redesign.
- *Driver redesign:* This is the redesigning of the software drivers' work in order to be consistent with hardware changes.
- *Interface modification:* Modify the soft IPs' communication bus interface protocol or add an interface converter.
- *Vendor restriction:* Vendor restriction is specified for an FPGA platform. It means that the FPGA devices can be changed, but this is restricted to the same vendor.
- *Vendor independent:* In contrast with vendor restriction, it does not have any restriction regarding the vendors. Any vendor device could be used.
- *Major system redesign:* It means that there is a whole redesign of the FPGA system, including the on-chip hardware and software driver redesign.
- *Minor system redesign:* Redesign parts of an FPGA system, including parts of the on-chip hardware and software driver redesign.

3.5 RESULT

Table 3-1 presents the results of the risk analysis for the design cases defined in section 3.1. The following is the explanation of table 3-1 for each of the risk scenarios:

- *Microprocessor obsolescence:* For a COTS product based platform, if the MCU becomes obsolete, the entire system will become obsolete. It results in a major board redesign and driver redesign for a new MCU system. While the FPGA microprocessor is described as a synthesizable soft code, such a special form completely eliminates the risk of microprocessor obsolescence. The legacy IP can still be implemented on a recent FPGA device, which can solve the component obsolete issue described in section 2.2.1.
- *Peripheral interface obsolescence:* The RS-232 serial interface in the system has the risk of obsolescence. If the MCU on the COTS platform does not support the

new peripheral interface standard, it should be replaced or a new interface controller should be integrated together with a physical interface circuit. Both situations will lead to a major board redesign. While for the FPGA platform, the interface controller can be changed by replacing a soft controller IP on-chip. The physical interface circuit (voltage converter etc.) and connector must also be changed which is a significantly easier task. A software driver redesign is necessary in order to adapt the new interface controller to the system. This will ease the peripheral interface obsolete issue of section 2.2.1.

- *Communication bus obsolescence:* For the COTS platform, the whole system has to be redesigned since every component which is associated with the bus is obsolete. The FPGA soft IP does not have any risk of obsolescence, but new IPs will face interface mismatch problems with an old communication bus. Interface modification is required if the new IPs are integrated into the system, which will ease the communication bus obsolescence describe in section 2.2.1.
- *Better circuit technology migration:* It is difficult for the COTS IC platform to benefit from better circuit technology requiring a major board and driver redesign. However, all the "soft" systems on the FPGA proposed in section V have the capability of accommodating new technology, only requiring minor board redesigns to redefine the pins for the new FPGA chip. However, a technology migration with regards to cases 2 and 4 is restricted to using the same vendor's device. Case 3 is the best choice regarding the maintenance issue described in section 2.2.2 because of its device independency.
- *FPGA Vendor device migration:* Vendor portability is an issue which is only relevant for FPGA systems. As has been mentioned previously, case 2 is a vendor specific solution and thus it is not possible to migrate it to other vendors' devices. If the vendor ceases to provide the devices, e.g. vendor bankruptcy, the only choice is to redesign a whole new system for another vendor's device. There is a better situation associated with case 4 as parts of the system are vendor independent and with the assistance of a new bus bridge, these parts could be migrated to other vendors' device. The design effort as compared to that for case 2 is significantly lower. For example, the Avalon to wishbone bus bridge IP could be used if the system is migrated to the Altera's device. Case 3 is a totally vendor independent system and thus offers the best portability from the FPGA platform cases and it would be easy to implement the vendor device migration.
- *Function change:* The function must be changed for any new function requirement or for removing bugs in the current component or system. It is a costly task for a COTS platform, because it results in a major board redesign. Due to FPGA's reconfigurabilty, the function in the form of a soft IP can be added, deleted or modified on-chip. In some situations, a physical circuit is required, such as an Ethernet controller. Some function changes do not even

require a board redesign but only a modification on-chip, if the FPGA is sufficiently large to accommodate the function, such as a video decoder.

Table 3-1. Consequences of different risk scenarios

Risks	Consequences	Case			
		1[a]	2[b]	3[c]	4[d]
Microprocessor obsolescence	Major board redesign	√	-	-	-
	Driver redesign	√	-	-	-
Peripheral interface obsolescence	Major board redesign	√	-	-	-
	Minor board redesign	-	√	√	√
	Driver redesign	√	√	√	√
Communication bus obsolescence	Major board redesign	√	-	-	-
	Interface modification	-	√	√	√
Better circuit technology migration	Major board redesign	√	-	-	-
	Minor board redesign	-	√	√	√
	Driver redesign	√	-	-	-
	Vendor restriction	N/A	√	-	√
	Vendor independent	N/A	-	√	-
FPGA vendor device migration	Major system redesign	N/A	√	-	-
	Minor system redesign	N/A	-	-	√
	Minor board redesign	N/A	√	√	√
Function change requirement	Major board redesign	√	-	-	-
	Minor board redesign	-	√	√	√
	Driver redesign	√	√	√	√

a. COTS IC based CAN controller system

b. Vendor specific FPGA system

c. Vendor and device independent FPGA system

d. Mixed FPGA system

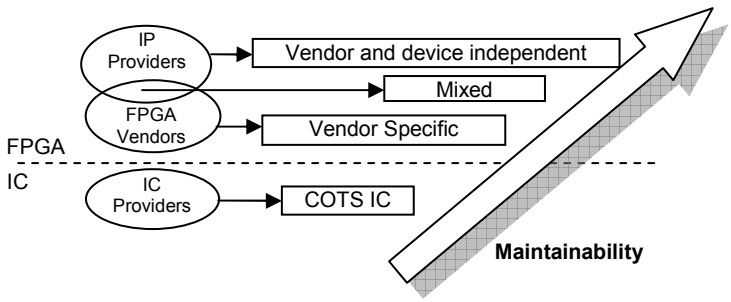

Figure 3-9. System maintainability model for different design technologies

3.6 MAINTAINABILITY FOR DIFFERENT DESIGN CASES

According to the evaluation in section 3.5, Fig. 3-9 can be used to encapsulate the conclusion for the case study.

3.6.1 COTS IC platform

COTS IC solution has no reconfigurability and portability. The MCU and CAN controller are in the fixed hard form provided by the IC providers. The platform defines its specific drivers and software applications. After the completion and release of the system design, it becomes difficult to make any further changes. Therefore, it always requires a major board and driver redesign as shown in table 3-1, the implication of which is that this will involve high maintenance costs. However, the benefits of the COTS IC are its mature technology and market. COTS IC design technology is now the dominant design method for an embedded system. Compared to the FPGA platform, it is more attractive with respect to higher performance, lower power consumption and cost.

3.6.2 Software issue for maintainability

Although software development and related tools are not included in the risk analysis, they also represent an unavoidable part in the present day embedded systems' development.

The software application of an embedded system is usually programmed in a high level language (e.g. C), which always contains some processor dependent code. Even if it is a non-processor-specific code, the device drivers and device management, initialization and locator modules and initial boot-up record data require modifications when the hardware changes [20]. The majority of the currently available software written for embedded systems is almost 100% target dependents. Any change to the hardware requires a change in the software [21]. If the microprocessor is replaced by a different one, the software has to be consistently modified. The development environment, such as the compiler and library, can also be different. Such software modification and environment change will result in a system re-verification, which is time consuming. Depending upon the volume of the code, a redesign can cost hundreds of man-years of time, much of which will be devoted to validation and testing [22]. For this issue, the portable code [12] is proposed, which allows compiled software to be executed on any platform without change thus reducing the cost of hardware obsolescence.

Using an embedded operating system can also mitigate the system migration problem. For example, embedded Linux brings vendor independence. Vendors of all embedded Linux distributions have more or less the same business model. The distributions are variations of the same theme. They all have the same and common basic components such as Linux kernel, libraries, basic utilities, and the like [23].

3.6.3 FPGA platform

The FPGA system is described using a high level language and is implemented on a single chip. If sufficient space is reserved on the chip, a system function could easily be changed using EDA tools. Sometimes, the function requires only small changes of the physical circuit and connectors as thus a minor board redesign, as is shown in table 3-1. Thus the embedded system built on the FPGA platform is very promising when maintenance issues are taken into consideration.

3.6.3.1 Vender specific system

Vendor specific design technology is possibly the most widely used on an FPGA platform. Provision of the IPs for the system is by the FPGA vendors. IPs, such as this, are of reliable quality and it is very easy to integrate them into the system. Designers can obtain adequate support and guarantees. The system might be portable within the same vendor's devices. However, such system design technology is unable to eliminate the risk that a vendor might cease to provide the devices (obsolescence). The reason for using a mixed solution is that it has better portability and is associated with the device independent parts. A designer can make an effort to integrate the OpenCores parts to other vendors systems by using a different bus bridge. Such mixed platforms benefit from vendor specific as well as vendor and device independent IPs, which can provide a compromised alternative for an embedded system design.

3.6.3.2 Vendor and device independent system

The FPGA platform implemented with vendor and device independent soft IPs is a preferred solution from the viewpoint of system maintenance, since it can solve all the maintenance problem issues described in section 2.2. The whole system, including the hardware and software, can be accommodated into any vendor device with any technology. The software application and GNU toolchain can be used on a new system device when performing the system migration. This thus eliminates the risk of device obsolescence from providers and it is also possible to benefit from the use of new circuit technology or lower cost hardware. Functions can also be modified for specific customer's requirements. Therefore, such a method could be compared to the traditional manner for a COTS IC based embedded system design technology and the system on an FPGA, in combination with soft IPs has significantly higher maintainability according to the results shown in table 3-1.

It is well known that FPGA's circuit technology and performance are not always growing as fast as a COTS IC component, but this could be tolerated by control, monitoring or communication systems, which are manufactured in low volumes [24]. Many of these systems do not require the newest state of the art

technology and would be in use, preferably, for more than 10 years, such as is the case for the CAN controller system.

The soft IP market is, at present, not sufficiently mature and for this OpenCores based platform, the quality of the open source IP is not satisfactory, since the designer can obtain very limited guarantee from providers. Because there is a lack of intensive verification, our experience is that IPs always contain bugs. However, it is to be expected that in the future, the high reliability vendor and device independent soft IP could be delivered by third party IP providers. Such IPs will become very good resources for designing a long life cycle embedded system.

3.7 CONCLUSION

A CAN controller system case study is present in this chapter. Various potential risks and their consequences for a long life cycle embedded system are discussed. It was proposed that an FPGA platform be used to replace the COTS IC platform for the system design. Different CAN controller system platforms have been evaluated using a number of risk scenarios. The results show that an FPGA platform with vendor and device independent soft IP has the highest maintainability.

4 FPGA IP PORTABILITY ANALYSIS

In last section, we proposed the use of an FPGA platform implemented with vendor and device independent soft IPs. This technique can increase the maintainability of an embedded system. In this section, the portability issue for an M-JPEG decoder will be analyzed and discussed.

4.1 INTELLECTUAL PROPERTY

Due to the rapid development of silicon technology, the capacity and performance of FPGAs has improved significantly every year. This allows the designers to build more complex SOCs. In a SOC design, a shorter product life cycle means a shorter time to profit from sales. It is widely recognized that the reuse and sharing of IPs is becoming fundamental to closing the deep sub-micron design gap for successful SOC design [25]. The semiconductor IP industry is over 15 years old but the reuse of IPs still contains many challenges for IP providers, system designers, IP business and IP tool developers [26].

A reusable IP [27] can be a digital IP or an analog/mixed-signal (AMS) IP which is in the form of a processor, memory, decoder, mixed-signal converter, etc. Digital IP is the most popular form for design reuse in present day industry [28]. They are divided into three categories: soft, firm and hard [29]. A soft IP is a hardware specification at the register transfer level (RTL) and this specification involves synthesizable code written in HDL. It is a more suitable form of digital IP, since HDL can be written in a technology-independent manner and synthesized to gate level. Its advantages include flexibility, portability and reusability [30]. A firm IP is in the form of a parameterized netlist and a hard IP is a technology-specific layout. For FPGA system design, the IP usually refers to either a soft or firm IP.

System designers usually obtain IPs, FPGA devices, library and tools from providers. In the majority of cases, they have a very limited knowledge with reference to the structural content of the adopted IP, so it must be considered to be a black box [30]. If an IP for FPGAs is truly portable, it must easily adapt to

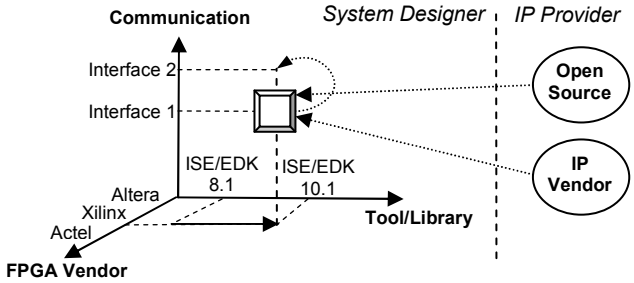

Figure 4-1. IP portability issue for system designers

different communication interfaces, being portable between different FPGA vendors and devices and having no dependencies with regards to the tool set used for the system design. This is illustrated in Fig. 4-1. Firm IPs for FPGAs are usually more or less technology dependent netlists which are also dependent on the used tool set.

4.2 PROJECT BACKGROUND

The aim of this work is to implement a real-time video processing system on an FPGA platform. The video system receives a video stream from several network cameras and manipulates them into a single video stream to be displayed on a single monitor. The system will be integrated into an industrial system of construction machinery. In order to develop the system, a host development computer, an FPGA board, a display monitor and network cameras are required. Fig. 4-2 depicts the real-time video processing experimental system. On the FPGA a video decoder, VGA controller and a soft microprocessor running Linux operating system were implemented. The host computer also acts as the console for the target board in relation to monitoring and recording the results through a UART interface or a telnet protocol.

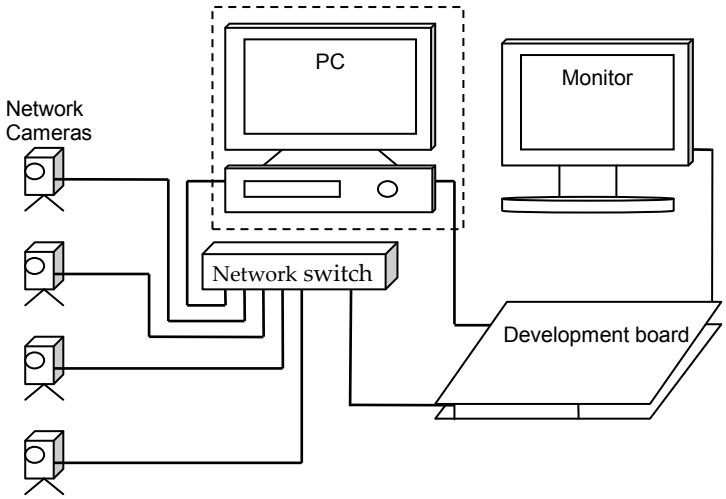

Figure 4-2. Architecture of the real-time display of multiple video streams

4.3 M-JPEG DECODER

JPEG, or Joint Photographic Experts Group, is a standardized image compression mechanism. The lossy compression method takes advantage of the visual capabilities of the human eye. This kind of file can be split into two areas, namely the headers and the compressed scan data. The headers contain the information about the compressed data (size, format and so on) as well as the

quantization tables and Huffman tables. This standard defines many different techniques used to compress, decompress and store image data. Whereas an M-JPEG is an informal name for multimedia formats, only the result of applying a JPEG to individual frames of a video sequence is called the M-JPEG.

M-JPEG compression format uses standard JPEG still images for video streaming. These images are displayed and updated at a rate which is sufficient to create a video stream. Although this method demands many bandwidths, it provides excellent image quality and can provide access to every individual image contained in the stream.

This project implemented an M-JPEG decoder project [31] that was a free download from OpenCores [32]. The decoder IP will be integrated into an industrial system of construction machinery which has long life cycle expectancy. It is used as a test case for the portability analysis described in section 4.4.

The M-JPEG decoder project was originally developed using ISE/EDK 8.1 on the Xilinx Virtex II pro FPGA. Fig. 4-3 shows the architecture of the decoder system. Three main blocks are involved: *M-JPEG Decoder Function*, *OPB IPIF* and *VGA Controller&Reorder*. The *Decoder IP* behaves as a master on the OPB enabling the decoder to retrieve compressed JPEG data from *Memory*.

OPB IPIF is the OPB interface protocol. The *M-JPEG Decoder Function* decodes the compressed JPEG data and outputs the uncompressed RGB data in a sequence corresponding to a series of Minimum Coded Units (MCU). This uncompressed RGB data is output directly to the *VGA Controller&Reorder* and further as an analog VGA signal to the *Monitor*.

The internal data flow of the *M-JPEG Decoder Function* is further illustrated in Fig. 4-4. Compressed data enters the decoder at the *Input Buffer* and decompressed data is output from *YCbCr2RGB* (YCbCr to RGB color converter). Check FF is a function to check the end of image (EOI) marker.

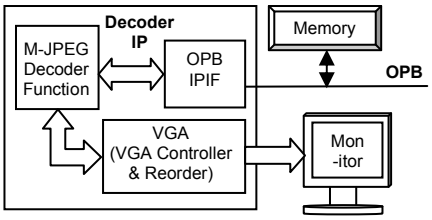

Figure 4-3. Block diagram of the decoder project

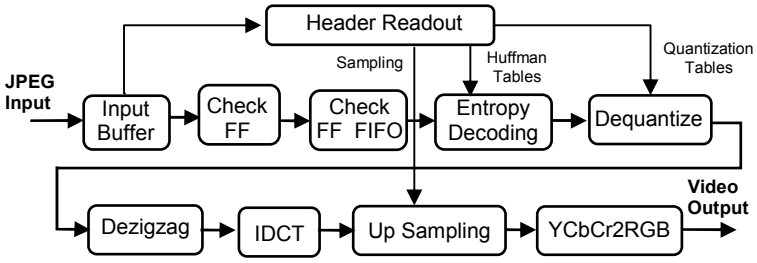

Figure 4-4. Internal data flow graph of the M-JPEG decoder

4.4 PORTABILITY ANALYSIS

This section describes the experimental method used to analyze the portability of the M-JPEG decoder IP described in section 4.3. This experiment includes all three dimensions shown in Fig. 4-1: Communication, FPGA Vendor and Tool/Library.

The portability was firstly analyzed with respect to the FPGA Vendor and Tool/Library. Consideration in this first part will be given to the decomposition of the hardware specification into a number of components shown in the data flow graph in Fig. 4-4. For each one of these components' corresponding to a hardware specification file, the following will occur:

- Investigate the type of HDL specification, Firm IP (technology dependent netlists), Soft IP (RTL specifications) or Soft&Firm (a mix of soft and firm specifications).
- Attempt to synthesize the hardware specification for Xilinx, Altera and Actel FPGA devices to verify whether or not this is possible.
- Transfer the hardware specification from the Xilinx tool set ISE 8.1 to the ISE 10.1. Test whether the transferred specification file can be synthesized in the ISE 10.1.

Secondly, the decoder IPs portability will be analyzed with respect to *Communication*. This analysis is to be conducted on the decoder's hardware architecture to determine whether it allows the communication interfaces to be easily modified. For an easy change of communication interface, it is important to verify that the hardware components are designed with a clear separation between communication and computation.

4.5 ANALYSIS RESULTS

This section describes the results from the case study described in section 4.4.

Table 4-1. Portability analysis with respect to FPGA vendor and tool/library

Component	Type of specification	Library instantiations	Portability	
			Tool/ library	*Ven- dor*
Input Buffer	Firm	FIFO	-	-
Check FF	Soft	-	√	√
Check FF FIFO	Firm	FIFO	-	-
Entropy Decoding	Soft & Firm	Memory, Shift Reg	-	-
Dequantize	Soft & Firm	Multiplier, Shift Reg	-	-
Dezigzag	Soft	-	√	√
IDCT	Soft & Firm	2-D DCT	-	-
Up Sampling	Soft & Firm	Block Memory	-	-
YCbCr2RGB	Soft	-	√	√
Header Readout	Soft	-	√	√
VGA Controller & Reorder	Soft & Firm	Block Memory, Digital clock manager(DCM)	-	-
OPB IPIF	Soft	-	√	√

4.5.1 Portability with respect to FPGA vendor and tool/library

Table 4-1 shows the library instantiations (column 3) in addition to a portability analysis with respect to *Tool/library* (column 4) and *Vendor* (column 5) for each component (column 1) in the *M-JPEG Decoder IP*. Column 2 shows that this IP is not a pure soft IP but contains several technology dependent firm IPs generated by the Xilinx CORE Generator. A check is made with regards to their portability from the Xilinx tool and library set ISE 8.1 to ISE 10.1. These components are also synthesized for Altera and Actel devices in order to check their portability between vendors.

4.5.2 Portability with respect to communication interfaces

The M-JPEG decoder IP shown in Fig. 4-3 has two communication interfaces:

- **Video Input:** Compressed data enters the decoder at the *Input Buffer*. See Fig. 4-4. This is an FIFO buffer which connects to the OPB bus interface protocol *OPB IPIF*. See Fig. 4-3.
- **Video Output:** Decompressed data is output from the *YCbCr2RGB* and is further written into the input buffer of the *VGA Controller&Reorder* for analog interfacing with the Monitor.

In accordance with the experimental method specified in section 4.4, there is a requirement to analyze whether the two interfaces have a clear separation between the communication and computation.

- **Video Input:** This interface has an FIFO, *Input Buffer* which clearly separates the communication specified in *OPB IPIF* from the computation within the *Decoder IP*. This will allow a system designer to easily exchange the OPB bus interface to an arbitrary bus interface.
- **Video Output:** This interface outputs uncompressed RGB data in a sequence corresponding to a series of MCU blocks. The sequence of pixels carried by the analog video signal must correspond to either the progressive video or to the interlaced video. This means that data must be reordered from MCU wise to line wise. This reordering is implemented in the *VGA Controller&Reorder* component. If the system designer would like to exchange the current VGA interface into High-Definition Multimedia Interface (HDMI) or any other arbitrary video interface, the reorder of the video data will be missing. This means that the *Video Output* interface cannot be exchanged as easily as the Video input interface.

4.6 SOFT-IP INTERFACE MODIFICATION METHODOLOGY

To make efficient reuse of the IP and its interfaces, an interface-based soft IP model is introduced. Based on this model, the Soft-IP Interface Modification Methodology (SIPIMM) is proposed to ease the communication interface mismatch problem.

4.6.1 IP verification

When the integrator obtain the IP, it is necessary to know its usage history. From a functional point of view, it is valuable to know whether or not the IP has been proven in silicon. This detailed information includes in which device or configuration it was used and what the success rate was and so on. This is useful information which will assist the integrator in avoiding unnecessary detours.

After this investigation and depending on its outcome, an ambitious functional verification of the IP core should be carried out. This step can, for example, be particularly important for an open-source IP, which offers very limited guarantees. Any test bench or verification methodology from the IP provider will ease this work. Otherwise, this task can be very difficult and time consuming, especially for configurable IPs.

4.6.2 Interface-based soft IP model

A soft IP core has the possibility of communicating with an external system. The on-chip system includes an on-chip bus, other system components or peripherals. The core can be an initiator (master) core or a target (slave) core depending on the implementation. For the interface-based soft IP model, several boundaries inside the core should be clarified, as shown in Fig. 4-5.

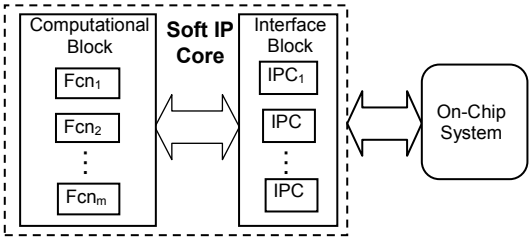

Figure 4-5. A generalized interface-based soft IP

1) *Isolate the computation logic from the interface logic:* The soft IP core can be mainly divided into two blocks: computational block and interface block. The *Computational Block* contains the main logic functions of an IP. The *Interface Block* implements all IP communication divided into one or several Interface Protocol Cells (*IPC*) as well as various ports for dataflow, control signals and debug signals. The *Computational Block* has a higher chance of being re-used since the *Interface Block* might be changed frequently due to different system configurations. This separation of computation from communication is effective for IP database management. It is a suggestion to the IP providers for creating reuse of IP in SOC Design [33].

2) *Boundaries between functions:* Different functions in the *Computational Block* should have clear boundaries (Fcn_1 to Fcn_m). It might occur that the integrators only require parts of the functions of a *Computational Block*. A reduced *Computational Block* must then be working correctly after the removal of some of the unnecessary functions.

3) *Boundaries between IPCs:* When using multiple interfaces, each *IPC* should be independent (IPC_1 to IPC_n). This also prepares the way for selective reuse in the future design.

4.6.3 Interface modification

1) *Interface selection:* Before designing the *Interface Block*, the integrator should already have conducted sufficient investigation in order to make a decision regarding which type of communication interface is the most suitable for the design specification. When the core requires multiple interfaces, the *Interface Block* is merely the assembly of different IPCs. *IPC* can be predefined and reused. This method for interface reuse can save a great deal of time since the designer need only to change some parameters without compromising the functionality of the IP core.

2) *Interface design:* One or more IPCs might be designed if they are not predefined.

- IPC design: *Computational Block* is a set of *Dummy Functions* in this case. See figure Fig 4-6. This set of dummies can be any synthesizable simple logic or test vector generator. A simulation of the IP that contains the

Dummy Functions and the *IPC* can be made using a simple test bench. Following this, the RTL code can be synthesized and the software application can be written. The IP verification can then be implemented in the FPGA prototype. Because the aim for the *IPC* for widespread reuse block, the aim in relation to the verification must be for zero-defect.

- IPCs assembling (multiple interfaces): Iteratively assemble and verify the designed or predefined IPCs one by one, until the whole *Interface Block* is verified through simulation and hardware prototyping. A model of an assembled *Interface Block* with *Dummy Functions* is shown as Fig 4-7.

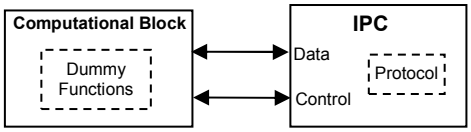

Figure 4-6. IPC design with a dummy function in the computation block

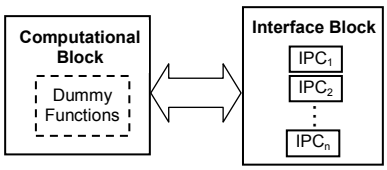

Figure 4-7. Interface design with dummy functions in the computational block

3) *Interface mapping:* Remove the original *Interface Block*. Map the output of the *Computation Block* to the input of the newly designed *Interface Block*. Write a test bench to verify the functionality of the complete IP as depicted in Fig. 4-5.

4.6.4 IP integration

Synthesize the RTL code. Write the software driver and test the software for a functional verification of the IP core. Integrate the entire IP into the system by connecting it to a simplified SOC for functional verification. Run the test software for the entire system. Evaluate the IP function and performance in order to finalize the design.

4.7 INTERFACES MODIFICATION OF M-JPEG DECODER

This section describes the work of modifying the M-JPEG decoder's interfaces described in section 4.3 using the proposed methodology SIPIMM.

4.7.1 IP verification

Firstly, the functionality of the original decoder IP was verified by simulation using the test bench supplied as part of the decoder project. Following this, we implemented the original system of the project on our FPGA board. The IP

functionality was verified using the original development environment of the project.

4.7.2 Interface-based soft IP model

The computation and interface boundaries were vague in the original IP structure. The *VGA Controller* (interface protocol) and the *Reorder* (computational) were blended. It represented a low portability and was difficult to migrate to another system with different functions and different bus interfaces. In order to integrate the core into our system and increase its portability, several modifications to its structure were performed.

For this work, we defined the interface-based soft IP model shown in Fig 4-8. The *Reorder* function was made into a part of the *Computational Block* and the *VGA* function was removed from the *Decoder IP*. The Interface Block contains two IPCs for the streaming of both compressed and uncompressed video. There is clearly a distinct separation of computation from communication in the developed interface-based soft IP model shown in Fig 4-5.

4.7.3 Interface modification

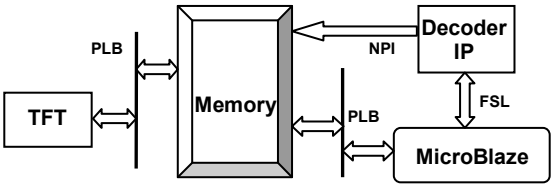

Figure 4-8. Block diagram of complete SOC including the updated decoder IP

1) *Interface selection:* According to the investigation and system specification, the Fast Simplex Link (FSL) and Native Port Interface (NPI) interfaces were selected to replace the OPB and VGA interfaces. The *On-Chip System* feeds the decoder IP with the compressed video through the FSL interface at IPC_1. The decoder IP outputs data to the system through the NPI interface at IPC_2. The NPI interface was selected for its low latency and high bandwidth target to memory which is ideal for real time video communication. FSL and NPI are custom interfaces and not predefined, so the two IPCs must be designed.

2) *Interface design:* The FSL and NPI interfaces were designed in accordance with Fig 4-6. The *Dummy Function* was designed as a clock counter for verification of both interfaces. After simulation and prototyping of both FSL and NPI, these interfaces were merged into a single interface block in accordance with Fig 4-7. The *Dummy Functions* were now selected as a copy of the input data from the FSL to output on the NPI.

3) *Interface mapping:* The reorder function component was connected to the M-JPEG decoder component output. The FSL interface was mapped as an input port to the decoder function. The NPI interface was mapped as an output port from the decoder function. Using a debugger, it was possible to verify an image that was decoded in the hardware and written to the system's memory through the NPI output.

4.7.4 IP integration

The *Decoder IP* shown in Fig 4-9 was integrated with a complete SOC consisting of a *MicroBlaze* soft processor core, *Memory* controller, a *TFT* controller for video displaying and other components not shown in the figure. The *MicroBlaze* processor feeds the *Decoder IP* with compressed video through the *FSL*. The decoded video is written directly to memory by the decoder through the *NPI* and into the same video memory area as is also read by the *TFT* controller on the *PLB*. The graphics displaying this SOC were visually verified by running the system on a prototyping board.

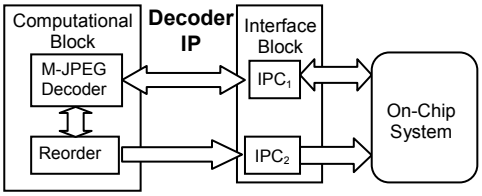

Figure 4-9. Interface-based soft IP model for an M-JPEG decoder

4.8 CONCLUSION

This chapter presents a case study analyzing the portability of an FPGA-based M-JPEG decoder IP. The use of vendor and tool dependent firmware specifications within the M-JPEG decoder is shown to limit the decoder's portability with respect to development tools and FPGA vendors. The lack of any clear separation between computation and communication is shown to limit the decoder's portability with respect to different communication interfaces. A methodology SIPIMM is proposed to increase the IP's portability regarding communication interface.

5 COMPONENT OBSOLESCENCE MANAGEMENT MODEL FOR LONG LIFE CYCLE EMBEDDED SYSTEM

This section discusses the component obsolescence problem and presents a mathematic model of obsolescence management for long life cycle embedded system maintenance. A simple CAN controller system case study is also shown in order to apply this model.

5.1 EOL SOLUTIONS

When components arrive at the EOL state, designers usually receive an obsolescence notice from the vendor. In this situation, a short term or long term strategy should be implemented to solve the obsolescence problem.

5.1.1 Lifetime buy and Last-Time-Buy

Lifetime buy is to buy components, which covers the entire system life. Last-time-buy (LTB) is to cover a short period of time as a short term strategy. [34]

The lifetime buy and LTB problem have two aspects: 1) demand forecasting, and 2) Order quantity determination.

Organizations that make lifetime buys or LTB of electronic components generally have little or no control over the supply chain for the components and cannot manufacture the components themselves. These organizations can purchase components until the manufacturer discontinues them at which time they must place a final order or implement some other mitigation strategy. [34]

The management strategy associated with lifetime buys of electronic components is to determine the number of components to purchase. Lifetime buys are risky, as forecasting demand and spares requirements for, potentially, 10-20 years into the future is not an exact science, especially in today's dynamic technology and market atmosphere. However, LTB only considers a short period of time to wait until a long time term strategy is taken. LTB is preferred in this paper, because it is less risky and easier for an organization to estimate the order quantity.

5.1.2 Redesign

Redesign is a long term strategy for obsolescence management. According to its complexity level, we could divide redesign into several categories:

Component(s) replacement

Component(s) replacement is the pin-for-pin replacement of an obsolete component(s) with the same function and pin compatible component(s). Replacement includes using COTS components, with a modern solid-state component in the same package type for a legacy obsolescence component [35]. For

example, some physical interface circuit or transmitter can be replaced directly. This component replacement process does not require any board redesign.

Part(s) redesign

Part(s) redesign is the practice of redesigning parts of or even the whole circuit board in the system. The redesigned part(s) contains new circuitry but uses a function and interface equivalent to that of the original part(s). There are two different design processes for parts redesign:

One is to replace obsolete components in the circuit with components having a similar function in a more advanced technology. For example, the microcontroller can be replaced by a new one which has more advanced technology and functions. Such replacement may involve board redesign and software re-writing.

The second method is to reverse engineer the part(s). It is not always the case that it is possible to find a replacement part in the market. If components with similar function do not exist, a reverse engineering should be implemented. This process analyzes an obsolete part's intended function, then redesigns a new part and makes it interface so that it is compatible with the system. This method assumes that an accurate specification is available or can be created. Engineers often find the original documentation incomplete, or even inaccurate. Many systems have components whose uses are undocumented, and the original designers are no longer available to provide assistance. Accurately extracting information from legacy systems is time consuming and costly. The lack of available information also drives up the cost of developing test benches and test vectors necessary for validating the design.

System(s) redesign

Usually, when one part in a system becomes obsolete, other interfacing components will also have a high obsolescence risk [35]. Moreover, a system has been developed as a single entity, with much interdependence between its hardware and software and thus replacement or modification is usually as difficult as designing an entire system. [12] Therefore in some cases, a redesign of the whole system should be implemented. This process includes redesigning hardware and software, which is costly and time consuming.

5.1.3 Others

Aftermarket

There are some aftermarket manufactures that provide a custom assembly of obsolete integrated circuits using an existing wafer and die. However, the unit cost will be much higher and only a few species of obsolete components can be provided.

Emulation

A very flexible replacement solution consists of emulating the obsolete microelectronic parts by means of programmable logic devices such as FPGAs [36]. The approach is particularly appealing as it is a general solution to the obsolescence problem of digital electronic parts: FPGAs can implement the behavior of practically any digital component, provided that a suitable model of the component's behavior is available [37].

However, this method requires huge engineering design efforts and costs, including qualification testing and certification to prove that the emulated version is equivalent to the original hardware in terms of functionality, performance and signal integrity.

5.2 MAINTENANCE MODEL

A mathematic model is built for maintenance cost analysis caused by component obsolescence.

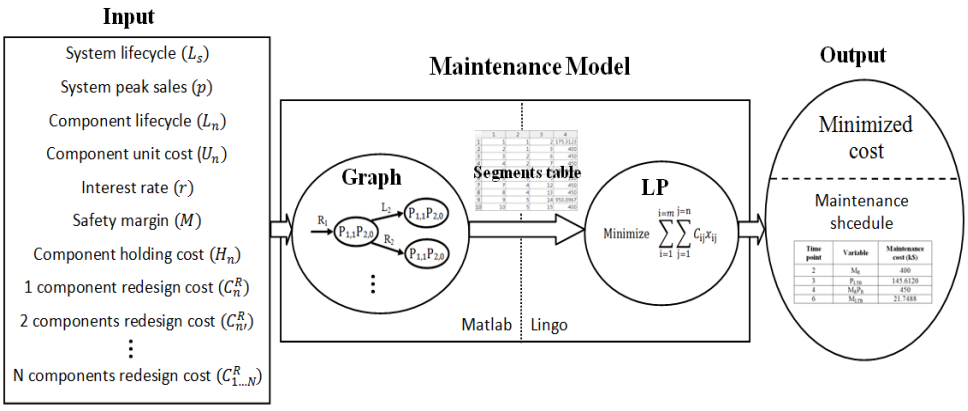

Figure 5-1. Maintenance model overview

5.2.1 Overview

An overview of the model is shown in Fig 5-1. Several input parameters should be given to the model. System life cycle expectation L_s is set at the beginning. Peak sales value p can be estimated. Commercial components database such as CAPS Expert from PartMiner can be used to obtain the product life cycle data and status. L_n represents the life cycle of component number n in the system, which can be obtained from the database. Unit cost, interest rate, safety margin and holding cost should be provided in order to estimate the LTB cost. The safety margin determines the extra order quantity percentage which been made in order to ensure the safety of sales. The holding cost refers to the spare components storage cost. The redesign cost is related to the component type and component

quantity. A multiple components redesign cost is less than the total redesigning cost for each component.

The model consists of two parts. The first part is to generate a graph in Matlab. A segments table is then output from Matlab for further optimization. Linear programming is used to calculate the minimized maintenance cost and maintenance schedule. More symbol explanations are now provided:

A = {1…N}, index of N components in the system

n ∈ A, component number in the system

n' ∈ {(x, y) | x ∈A, y ∈A, x≠y}, two components number in the system

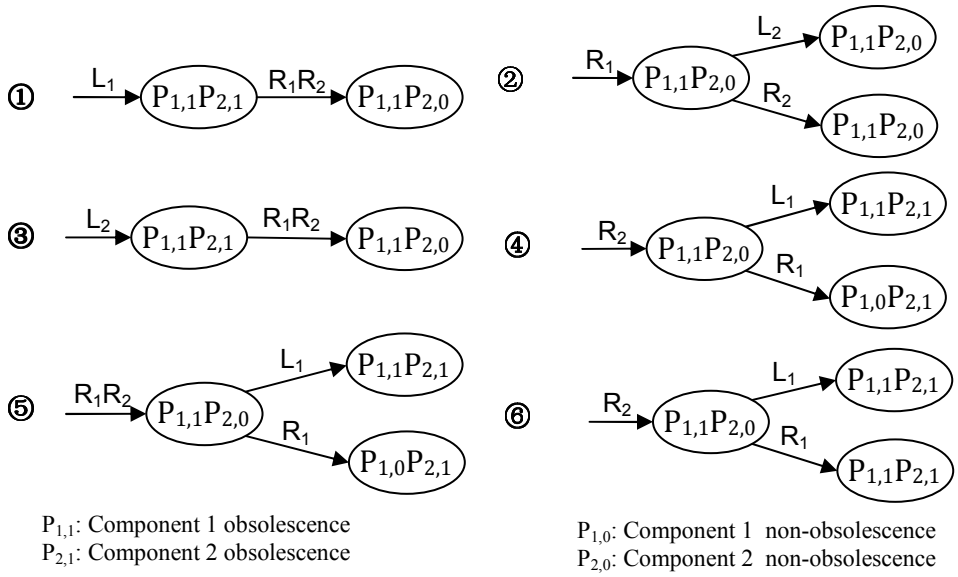

Figure 5-2. State transfer diagram

5.2.2 State transfer diagram

The model contains a graph part which is in the form of state transfer diagram. At the present time the model has a maximum components quantity limitation of 2, but it can be expanded to n components in future work. A two components state transfer diagram is shown in Fig 5-2. Suppose $L_1 < L_2$. The following provides the symbols explanation in the diagram:

$P_{n,s}$ represent a component status.

s ∈ {0, 1}, 0 represent non-obsolescence, 1 represent obsolescence.

Each component has two states in the model. $P_{1,1}$ means component 1 obsoletes while $P_{1,0}$ means component 1 will not become obsolete in a short period (non-obsolescence). The same representation is given for component 2 as $P_{2,1}$ and $P_{2,0}$. When one or several components become obsolete, an obsolete solution (e.g. R_1R_2 above the arrow) has to be implemented. Once the solution is taken, one state at the current time point will be transfer to another state on next time point. The arrow shows the state transfer along the time line.

For example, in the state transfer diagram ① of Fig 5-2, component 1 makes an LTB (L_1) in order to wait until component 2 becomes obsolete ($P_{1,1}P_{2,1}$). When both components are obsolete, a redesign involves two components which must be implemented (R_1R_2). After redesign, both components restart a new life cycle and component 2 will become obsolete first ($P_{1,1}P_{2,0}$) since L_1 is smaller. This is similar for the remainder of the state transfer diagrams and a total of 6 kinds of state transfer styles are shown in the Fig 5-2. ⑥ is a special case of ④, which denotes the time interval from state $P_{1,1}P_{2,1}$ to state $P_{1,1}P_{2,1}$ is equal to L_1.

In order to formulate a full binary tree for the model, the state transfer path for $P_{1,1}P_{2,1}$ to $P_{1,1}P_{2,0}$ is duplicated, shown in Fig 5-3. The path duplication will not affect the result, but it will make the graph simpler to solve by means of tools.

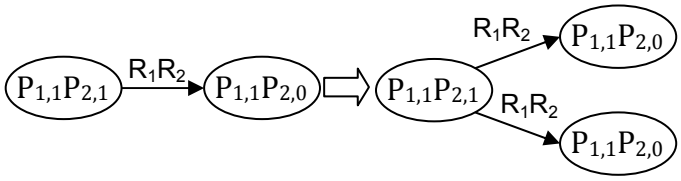

Figure 5-3. State transfer path duplication

The full tree graph information will be stored in a segments table generated by Matlab. The information in the table includes the start node number, end node number and cost for each state transfer path.

5.2.3 Cost estimation

Two methods are considered for obsolescence management in the model:

- Last-time-buy: This method is to conserve a safety stock of obsolescence components for a short period of time. LTB is a short term strategy and it will not renew the component life cycle. The main purpose is to delay the redesign time until any other component in the system becomes obsolete. The LTB cost depends on the sales value, interest rate, unit cost, holding cost and the safety margin as an uncertainty factor. The LTB order quantity is estimated

using the demand (D) for a certain period of time. ΔT represents the time interval between two obsolescence management time points.

$$D = \int_{T}^{T+\Delta T} f(x)\,dx = \int_{T}^{T+\Delta T} pe^{(-(x-\mu)^2/2\sigma^2)}dx \quad (5.1)$$

The LTB cost consists of:

Safety margin cost: A safety margin (M) of an order should be defined for component n. M is defined as a percentage of the estimated sales.

$$C_n^M = D * M * U_n \quad (5.2)$$

Interest rate cost: When a company makes an order for a component, the money spent for the order will generate an extra cost for component n caused by the interest rate.

$$C_n^r = (D * (1 + M) * r * \Delta T) * U_n \quad (5.3)$$

Holding cost: The LTB order quantity is always much larger than the normal order. A storage cost should be considered. H_n represents the average holding cost of component n in one unit of time.

$$C_n^H = H_n * (1 + M) * D * \Delta T \quad (5.4)$$

The total LTB cost can be estimated as:

$$C_n^L = C_n^M + C_n^r + C_n^H \quad (5.5)$$

- *Redesign*: Redesign is a long term strategy. It could be as easy as an electrical component replacement or as difficult as a system redesign including both hardware and software. The redesign cost depends on the number and type of components.

5.2.4 Linear programming

Linear programming is used to find the minimized total cost for the system life cycle maintenance.

Finding the lowest cost between the first node and the terminal nodes in the model can be viewed as a transshipment problem [38]. A ghost node is assumed, since it makes modeling easier. This node connects to every terminal node which is located at the system lifecycle terminal time point. The path cost between the ghost node and the terminal nodes is set to 0. The model can then be viewed as finding

the shortest path of sending one unit from the first node to the ghost node, with all other nodes in the model being transshipment points.

$$\text{Minimize} \sum_{i=1}^{i=k} \sum_{j=1}^{j=m} C_{ij} x_{ij}$$

Subject to

$$\sum_j x_{ij} = 1 \ (i = 1)$$

$$\sum_j x_{ij} = \sum_j x_{ji} \ (i = 2,3 \ldots, N_P)$$

$$\sum_j x_{ji} = 1 \ (i = N_P + 1)$$

i,j are the nodes number in the segments table.
k,m are the max start and end nodes number in the segments table.
C_{ij} is the path cost between node i and node j in the graph.
Variable x_{ij}=1, if the path from node i to j is chosen for shortest path, otherwise 0.

Lingo [39] is a comprehensive modeling language and a set of solvers for linear, non-linear, and integer programming from Lindo system Inc. The transshipment model is built and optimized by Lingo. The minimized total maintenance cost and optimized time schedule output from Lingo are given as the result.

5.3 CASE STUDY

In this section, a CAN controller industrial system is used as a case study to validate and analyze the model.

5.3.1 Experimental system

CAN is an industrial bus standard designed to allow microcontrollers and devices to communicate with each other. This project started from a CAN controller system used in industrial construction machinery. The basic CAN node contains an MCU or microprocessor and a CAN controller. In this case, cost, performance and power consumption are not critical issues. However, this type of long life cycle system requires greater consideration to be given to maintenance.

For our experimental CAN controller system, an MCU (PIC16F876) was implemented onto the board. A COTS CAN controller (MCP25020) was implemented on the board communicates with the MCU via Serial Peripheral Interface (SPI) bus. The CAN node architecture can be seen in Fig 5-4. For simplicity, other physical circuits are not considered.

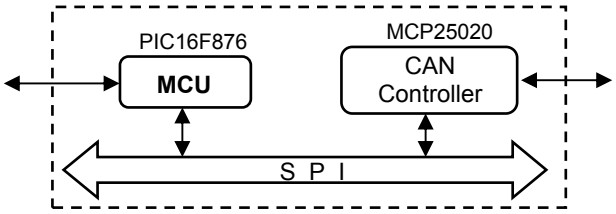

Figure 5-4. CAN controller system architecture

5.3.2 Model validation experiment

MCU is considered as component 1 and the CAN controller considered as component 2. The life cycle of MCU is 2 units of time and the CAN controller is 3 units of time. The system life cycle is 7 units of time.

The sales value of the system can be estimated from formula (1). In this case the mean μ is equal to 3.5 units of time. The standard deviation σ is equal to 1 unit of time. The sales peak p is assumed to be 5,000. The LTB cost can be estimated by means of formula (6). The interest rate (r) is given as 10%. Safety margin is set to 20%. Holding cost for both components (H_1 and H_2) is given as 2 for every unit of time. The cost value is represented by $. The given parameter values are synthetic and experimentally selected.

Table 5-1. Input parameter

Parameter	Value	Parameter	Value
L_s	7	r	10%
p	5,000	M	20%
L_1	2	H_1, H_2	2
L_2	3	C_1^R	40,000
U_1	30	C_2^R	15,000
U_2	10	$C_{1,2}^R$	45,000

5.3.3 Model response analysis

More experiments are implemented in order to analyze the response to different input parameter values.

Some parameters such as interest rate and, storage cost will affect the LTB cost for both components. A test of maintenance cost responding to the interest rate is implemented. The interest rate is supposed to increase from 0 to 100% (100% should not occur in the real world and it is merely an experiment).

Another test is to increase the unit cost of the CAN controller (U_2) from 10 to 15 and 20, in order to analyze how the maintenance cost is responding to the unit cost.

5.3.4 Result

The state transfer graph for the whole system life cycle is shown in Fig 5-5. The lifecycle curves of the system and components are also integrated into the graph. The curves provide the sales value and lifecycle for the components and system. In relation to the algorithm, some states nodes will be located at time 8. Since the system lifecycle is 7, the management work should be ended at time 7. The nodes at time 8 (dotted circle in Fig 5-5) should be relocated to time 7. The algorithm for this terminal time point situation is applied in the model. The final graph then has terminal time points (24 to 31) at time 7. There are 31 total possible states nodes which exist in the system life cycle.

Matlab is used to generate the graph. The segments table will be output to Lingo for further optimization.

In this case study, a ghost node 32 is assumed. This node connects to every terminal node with zero path cost, as shown in Fig 5-6. The model can then be viewed as finding the shortest path of sending 1 unit from node 1 to node 32. The transshipment model is then able to be programmed and optimized in Lingo.

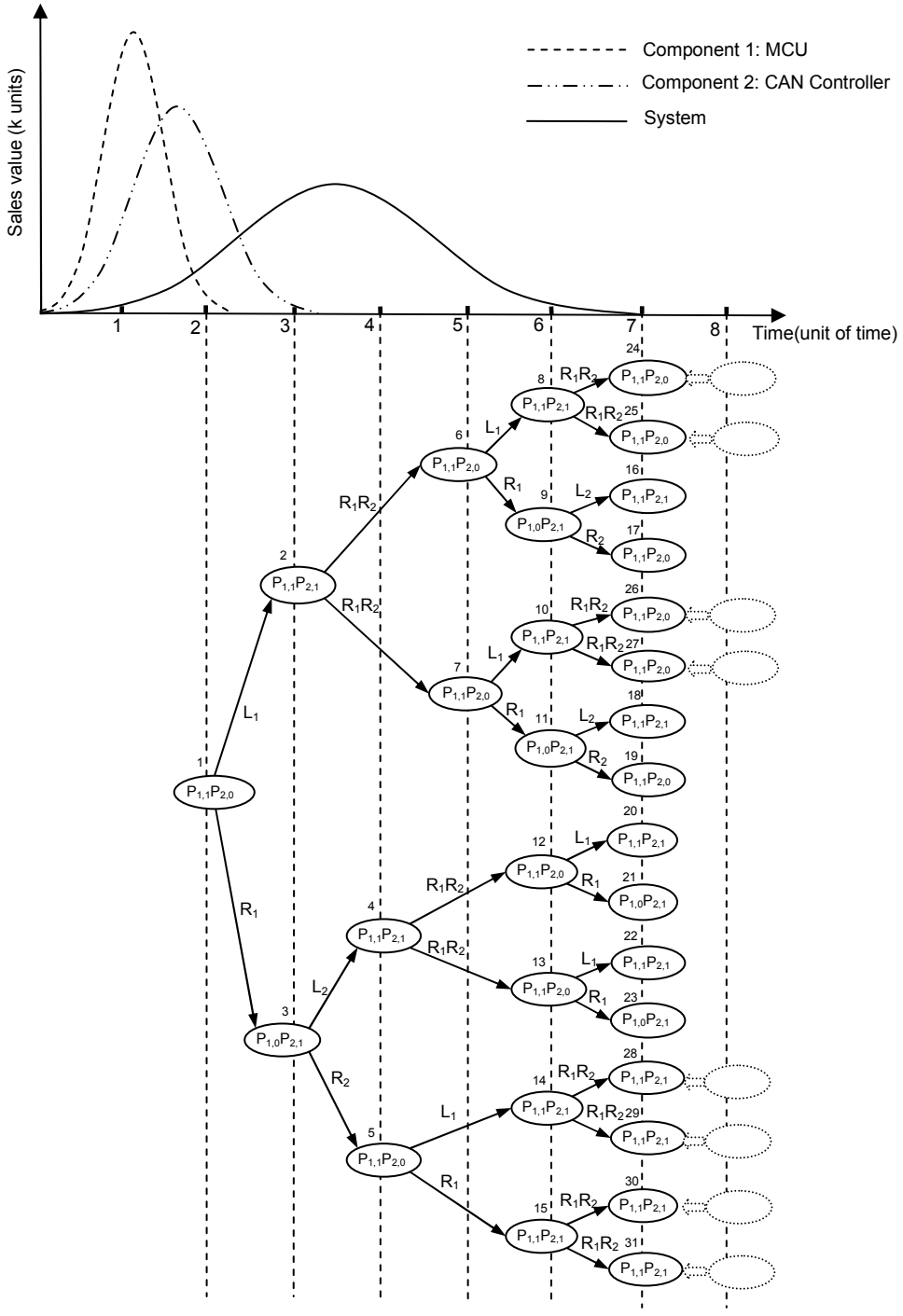

Figure 5-5. CAN controller system state transfer graph generated by Matlab

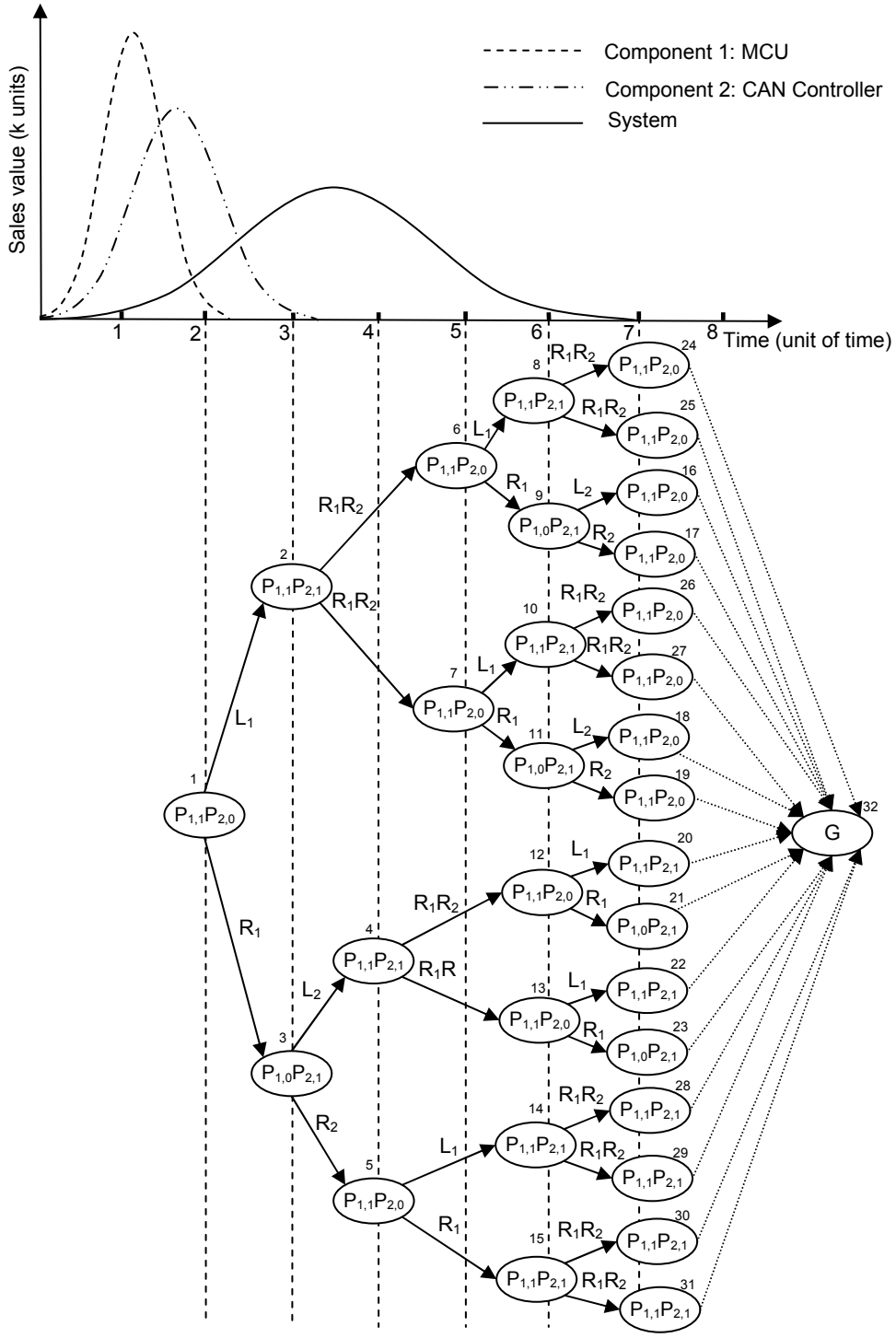

Figure 5-6. Final CAN controller system state transfer graph

The minimized system maintenance cost is estimated as 112.77k$. The optimized maintenance schedule is shown in Table 5-2. These results are based on the synthetic and experimentally selected parameter values in Table 5-1.

Table 5-2. Maintenance schedule

Time point	Variable	Maintenance cost (k$)
2	R_1	40
3	L_2	26.876
4	R_1R_2	45
6	L_1	0.898

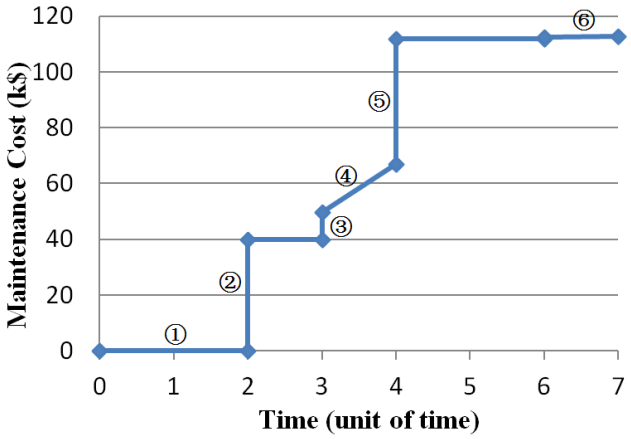

Figure 5-7. Maintenance cost VS Time curve

An explanation of the maintenance cost VS Time curve:

① No component obsolescence occurs. Maintenance cost will be 0 in this period.

② MCU obsolete at time point 2. An MCU redesign will be implemented and the cost will be 40k$. (It is supposed that the redesign time is negligible when compared to the system life cycle).

③ At time point 3, the CAN controller becomes obsolete. An LTB of a peripheral is implemented. The safety margin cost is 9598.504$ in this case and it will be generated at the time of making the LTB order.

④ In this period, the holding costs and the interest rate costs cause the maintenance cost to be linearly increased.

⑤ Both components are obsolete. A redesign involving two components is implemented and the cost is 45k$.

⑥ At time point 6, an MCU LTB is implemented. Since the time period is almost at the EOL of the system, the order amount will be minimal. The LTB cost is 0.9888 k$ in this case.

The maintenance cost responding to the interest rate curve is shown in Fig 5-8. The maintenance cost will be increased together with the interest rate until it is equal to 64.6%. The maintenance cost will remain at a constant value of 1.4*105 when the interest rate is larger than 64.6%.

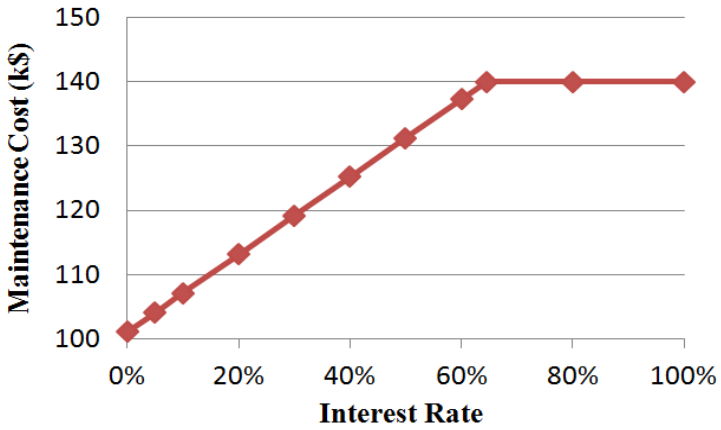

Figure 5-8. Maintenance cost VS Interest rate curve

From Fig 5-8 and the time schedule output from the model, it was determined that the model will prefer an LTB at two time points when the interest rate is low. The maintenance cost increases linearly when the interest rate is growing from 0 to 64.6%, according to formulae (4) and (6). When the interest rate is larger than 64.6%, the model will prefer to implement a redesign instead of an LTB, since the LTB cost exceeds the redesign cost. The redesign cost is not affected by the interest rate, so the maintenance cost remains constant.

Another experiment is to increase the unit cost U_2 from 10 to 15 and 20. The maintenance schedule comparison is presented in Table 5-3. The graph for the maintenance cost together with the sales curve is shown in Fig 5-9.

Table 5-3. Maintenance schedule for different unit price

Time point	U_2=10	U_2=15	U_2=20
2	R_1	R_1	L_1
3	L_2	L_2	R_1R_2
4	R_1R_2	R_1R_2	
5			R_1
6	L_1	L_1	L_2

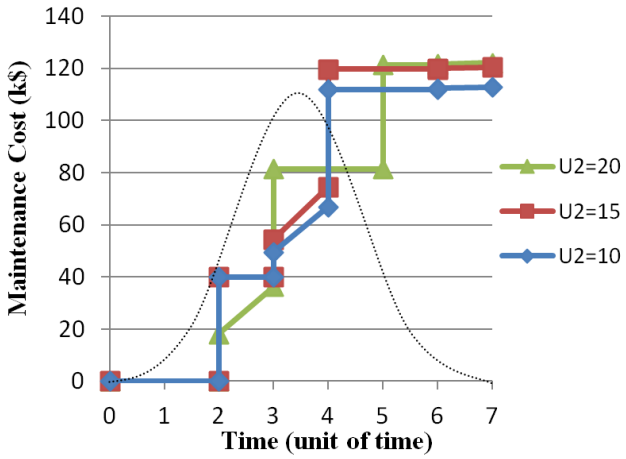

Figure 5-9. Maintenance cost VS Time comparison curve

Table 5-3 shows that the model will change the maintenance schedule when U_2 is 20. When U_2 is increased from 10 to 20, the LTB cost will be higher especially when the sales value is high. The increased unit cost will result in an increased interest rate cost and, safety margin cost, so the LTB cost will be increased according to formula (6). The model then plans to involve the LTB at an earlier stage and implements the redesign instead. This occurs when the system has a high sales value. Therefore, the total maintenance cost will not increase significantly. The implication is that the model will prefer a redesign instead of an LTB when the sales value is high, which meets our expectations.

At the present time the model that has been built contains many simplifications and has a limitation of 2 components. The experiment is implemented with assumed parameters values. The uncertainties for a real world application are also not considered. However, this is an initial step before it is expanded and it is verify using real industry statistical data.

5.4 CONCLUSION

This chapter discusses the component obsolescence problem and presents a mathematic model of obsolescence management for long life cycle embedded system maintenance. The model presented in this paper considers redesign and LTB as two management methods. This model can estimate the minimized management costs for different system architectures. A simple CAN controller system case study is shown in order to apply the model. A minimized management cost and an optimized management time schedule are given as the result. Test experiments of the maintenance cost responding to the interest rate and

unit cost are implemented. The responses from the experiments meet our expectations.

6 THESIS SUMMARY

The maintainability of a long life cycle embedded system for different design technologies has been analyzed. An industrial CAN controller system is used as an experimental system, which is implemented using different design technologies, generating four different cases. The essence of each case and the consequences associated with the different risk scenarios during system maintenance are analyzed.

Various potential risks and their consequences during product maintenance work have been discussed. This is of particular relevance for an embedded system which has long life cycle expectancy. The proposal was that an FPGA platform could be used to replace the COTS IC platform for the system design. Different CAN controller system cases have been evaluated using a number of risk scenarios.

The maintenance issue associated with the portability of the IP for the FPGA embedded system is highlighted. This has been conducted by means of a case study of an FPGA based M-JPEG decoder project from OpenCores. This case study analyses the IP's portability between communication interfaces, development environments and FPGA vendors. The analysis of the results from the case study allows us to identify the design techniques that offer the highest portability for an FPGA based IP. If designers use an IP in an improper way, they will face significant difficulties in maintaining a long life cycle FPGA embedded system.

A methodology for the modification of soft IP interfaces is proposed. This methodology is based on the fact that computation is divided into functional units which have a distinct separation from communication. The benefit is, clearly, the increased communication portability for a soft IP. In addition, the increased reuse of design works leads to a reduced work load for the IP integrator. A case study is presented to show that this methodology can be efficiently applied to a real-world design. There are many essential issues for IP reuse apart from that of an interface mismatch. IP design is a mixed topic with technical, financial and legal issues.

A mathematic model for long life cycle embedded system maintenance is presented. The model can estimate system maintenance costs for different architectures. This model can find optimized solutions regarding the components obsolescence problem. The minimized maintenance cost together with an optimized maintenance schedule can be estimated. An industrial CAN controller system is used as our experimental system.

6.1 CONCLUSION

The proposed system on FPGA in combination with soft IPs will increase the maintainability of a long life cycle embedded system. An FPGA platform with vendor and device independent soft IP is the preferred design technology. The result of our research could prove to be useful for designers who might be facing difficulties in relation to maintaining a long life cycle embedded system.

It is convenient to use the technology and tool dependent firm IPs generated by the FPGA vendor's design tools for both quick and easy design work. However, it is clearly shown in the thesis that this design technique should be avoided if portability and maintainability are more important than development time. Clear separation between the communication protocols and the IP's computational parts is another well known design technique that leads to higher portability and maintainability for an IP component.

The SIPIMM is proposed to ease the soft IP interface problem. The portability and reusability will be increased by using the methodology.

The presented obsolescence management model can offer maintenance strategy guidance to the designers who are faced with components obsolescence, although, at the present time, it has been built based on a number of assumptions and simplifications.

6.2 FUTURE WORKS

The maintainability analysis for a long life cycle embedded system will be extended for later PhD thesis work.

The number of IP components to be analyzed will be extended as well as for more FPGA vendors and development tools/library. A deeper analysis in relation to a number of issues regarding IP portability will be made. Finally, the idea will be refined to provide a more general theory for the IP portability issue. The IP interface modification methodology of SIPIMM should be practiced for more soft IPs, as well as being extended to the implementation of more vendor devices other than Xilinx.

The plan is to perform experiments with real life industrial statistical data and to expand the component quantity to be more generic for the mathematic maintenance model. A maintainability evaluation for different system design technologies including COTS and FPGA is also planned.

7 SUMMARY OF PUBLICATIONS

7.1 PAPER I

This paper presents and raises the issues of maintaining an embedded system which has a long life cycle. Such issues include obsolescence, function change requirement or technology migration etc. Systems' maintainability for different design technologies are analyzed. Different platform cases are evaluated by analyzing the technology of each case and the consequences of different risk scenarios during system maintenance. The result shows that the FPGA platform with vendor and device independent soft IP has the highest maintainability.

7.2 PAPER II

This paper suggests a soft IP interface modification methodology (SIPIMM) for systems on FPGA. SIPIMM targets an interface-based soft IP model which is introduced to ease the interface modification and interface reuse. The portability will be increased accordingly, so this will ease the maintenance issue for a long life cycle embedded system. A case study of an open-source IP is presented using SIPIMM for system integration.

7.3 PAPER III

This paper discusses the use of an FPGA IP to ease the problem of electronic components becoming obsolete. System migration between devices is unavoidable, especially for long life cycle embedded systems, so IP portability becomes an important issue for system maintenance. A case study is shown to analyze the portability of an FPGA-based M-JPEG decoder IP. The lack of any clear separation between computation and communication is shown to limit the decoder´s portability with respect to different communication interfaces. The use of technology and tool dependent firmware specifications within the M-JPEG decoder is shown to limit the decoder's portability with respect to development tools and FPGA vendors.

7.4 PAPER IV

This paper makes a deeper analysis of the component obsolescence problem and presents a mathematic model for a lifecycle analysis the long life cycle embedded system. This model can estimate the minimum maintenance cost for different system architectures. A simple CAN controller system case study is shown to apply this model. A minimum maintenance cost and an optimum maintenance time schedule are provided as the result.

7.5 AUTHORS CONTRIBUTIONS

The exact contributions of the authors of the four central papers in this thesis are summarized in Table 7-1.

Table 7-1. Authors' contributions

Paper #	Main authors	Co-authors	Contributions
I	XM	BT NL	XM : Analysed the maintenance issues. BT : Supervised the work and help to build risk analysis NL : Supervised the work
II	XM	BT NL	XM : Designed the methodology BT : Supervised the work NL : Supervised the work
III	XM	BT NL	XM : Made analysis and experiment BT : Supervised the work and help to made analysis NL : Supervised the work
IV	XM	BT LO	XM : Designed the model and implemented experiments BT : Supervised the work and gave suggestions LO : Helped to build linear programming part
1. Xiaozhou Meng (XM) 2. Benny Thörnberg (BT) 4. Najeem Lawal (NL) 5. Leif Olsson (LO)			

8 REFERENCES

[1] Steve Heath, Embedded Systems Design, 2nd edition, Elsevier Science. 2003

[2] Wayne Wolf, Computers as components: principles of embedded computing system design. 2nd edition, Morgan Kaufmann publishers. 2008.

[3] Raj Kamal "Embedded Systems: Architecture, Programming, and Design" McGraw-Hill, 2008.

[4] Solomon R., Sandborn P.A., Pecht M.G., "Electronic Part Life Cycle Concepts and Obsolescence Forecasting", IEEE Transactions on Components and Packaging Technologies, Vol: 23, 2000.

[5] Product Life Cycle Data Model, American Standard ANSI/EIA-724, Sept. 19, 1997.

[6] M.G. Pecht, D. Das, "Electronic part life cycle" IEEE Transactions on Components and Packaging Technologies, Mar 2000, pp. 190 - 192.

[7] PartMiner WorldWide Inc., https://www.partminer.com/

[8] P.A. Sandborn, F. Mauro, R. Knox, "A Data Mining Based Approach to Electronic Part Obsolescence Forecasting" IEEE Transactions on Components and Packaging Technologies, Sept. 2007, pp. 397 - 401

[9] P. Sandborn, "Trapped on Technology's Edge," IEEE Spectrum, 2008, pp. 42–58.

[10] PLB v3.4 and OPB to PLB v4.6 System and Core Migration User Guide. http://www.xilinx.com/support/documentation/sw_manuals/edk10_mg_ug.pdf

[11] "ElectronicsTalk," Aug. 19, 2003. http://www.electronicstalk.com/news/tex/tex489.html, Texas Instruments. Obsolescence Policy Gains Period of Grace.

[12] N. Audsley, I. Bate, A. Grigg, "Portable code: reducing the cost of obsolescence in embedded systems," Computing & Control Engineering Journal, 1999, pp. 98–104.

[13] J. Schmidt, E.F. Hitt, "Technology obsolescence (TO) impact on future costs" 17th AIAA/IEEE/SAE Digital Avionics Systems Conference, 1998. Pp. A33 - 1-7 vol.1.

[14] V. K. Madisetti, "Reegineering digital systems," IEEE Design & Test of Computers, 1999, pp. 15–16.

[15] CAN Specification version 2.0, 1991, Robert Bosch GmbH, http://www.semiconductors.bosch.de/media/pdf/canliteratur/can2spec.pdf

[16] MicroBlaze Processor Reference Guide, http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/mb_ref_guide.pdf

[17] LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a), http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf

[18] OpenRISC GNU toolchains, http://opencores.org/openrisc,gnu_toolchain

[19] Digilent Nexys™2 Spartan-3E FPGA Board, http://digilent.org/Products/Detail.cfm?Prod=NEXYS2

[20] Raj Kamal, Embedded Systems: Architecture, Programming, and Design. McGraw-Hill publisher, 1st edition, 2008.

[21] S.E.-D.A. Khalil, A.M. Wahba, "Usage of run-time re-configuration for system porting in automotive applications" 4th International Design and Test Workshop (IDT), Nov. 2009. pp. 1 - 6

[22] K. Parnell, "Driver assistance systems - real time processing solutions" Proceedings of IEEE Intelligent Vehicles Symposium, June 2003, pp. 547 – 551.

[23] P.Raghavan, Amol Lad, Sriram Neelakandan, Embedded Linux System Design and Development, Auerbach Publications, 2006, page 7

[24] J. Torresen, T.A. Lovland., "Parts Obsolescence Challenges for the Electronics Industry," IEEE Design and Diagnostics of Electronic Circuits and Systems, 2007. pp 1.
[25] N. Bray, "Designing for the IP supermarket" Fall VIUF Workshop, Oct 1999. pp. 8 - 13
[26] D. D. Gajski, et al., "Essential issues for IP reuse," Asia and South Pacific Design Automation Conference. Yokohama, 2000, pp. 37-42,.
[27] D. K. Kim, K. W. Kwon, J. C. Choi, C. D. Lee, "Reusable intellectual property cores in PC data protection ASIC design," The First IEEE Asia Pacific Conference on ASICs. Seoul, 1999, pp. 278-281.
[28] A. Vorg, M. Radetzki, W. Rosenstiel, "Measurement of IP qualification costs and benefits," Proceedings of Design, Automation and Test in Europe Conference and Exhibition. 2004, vol. 2, pp. 996-1001.
[29] M. Keating, P. Bricaud, Reuse Methodology Manual: For System-on-a-Chip Designs, 3rd ed., MA:Kluwer, Boston, 2002.
[30] R. Saleh et al, "System-on-Chip: Reuse and Integration," Proceedings of the IEEE. 2006, vol.94, pp. 1050-1069
[31] (M)JPEG Decoder, http://www.opencores.org/project,mjpeg-decoder
[32] OpenCores, http://www.opencores.org
[33] P.J. Bricaud, "IP reuse creation for system-on-a-chip design" Proceedings of the IEEE Custom Integrated Circuits,May 1999, pp. 395 – 401
[34] D. Feng, P. Singh, P. Sandborn, "Lifetime Buy Optimization to Minimize Lifecycle Cost" Proceedings of the 2007 Aging Aircraft Conference, Apr 2007
[35] R.C.Stogdill, "Dealing with obsolete parts" IEEE Design & Test of Computers, Aug 2002, pp. 17 – 25.
[36] L. Anghel, et al. "Preliminary Validation of an Approach Dealing with Processor obsolescence" Proceedings of 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Dec 2003, pp. 493 - 500.
[37] F. Abate, M. Violante, "Coping with obsolescence of processor cores in critical applications" IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, Oct. 2008. pp. 24 – 32
[38] Wayne L. Winston, Operations Research: Applications and Algorithm, 4th ed. Duxbury Press, 2003.
[39] Anon, Lingo, The Modeling Language and Optimiser, Lindo Systems Inc., Chicago, USA , 2003.