Open Source Hardware

by

Roberto Acosta
Bachelor of Science, Electrical Engineering
University of New Hampshire, Durham New Hampshire

Submitted to the System Design and Management Program in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Engineering and Management
at the
Massachusetts Institute of Technology
June 2009

Signature of Author [illegible]
Roberto Acosta
System Design and Management Program
May 2009

Certified by [illegible]
Eric von Hippel
Thesis Supervisor
Sloan School of Management

Certified by [illegible]
Patrick Hale
Director
System Design and Management Program

This page is left intentionally blank.

Open Source Hardware

by

Roberto Acosta

Submitted to the System Design and Management
Program on May 9, 2009 in Partial Fulfillment of
the Requirements for the Degree of Master of
Science in Engineering and Management

Abstract

Open source software development models have created some of the most innovative tools and companies in the industry today modifying the way value is created and businesses developed. The purpose of this thesis is to analyze open source hardware in its current state and its potential impact at several stages of the value chain.

Existing examples of open source hardware at different stages of the value chain are analyzed in terms of their innovation and potential impact to existing players in the value chain. An Ethernet framer is develop through the use of traditional development and benchmarked against a design developed based on open source hardware cores.

The research concludes with an examination of business models established around open source hardware.

Thesis supervisor: Eric Von Hippel
Title: Professor of Management of Innovation and Head of the Innovation and Entrepreneurship Group

Biographical Note

Roberto Acosta enjoys the challenges of building products. Roberto has been architecting, designing and building hardware for close to fifteen years. He has developed hardware for a wide range of applications and systems including but not limited to space electronics, handheld devices and telecommunication platforms.

Roberto Acosta joined the System Design and Management (SDM) program to increase his knowledge of product development. The skills gained from the program have taught him many of the important tools and approaches to architecting and designing complex systems. Roberto has also gained the skills required to bridge the gap between business strategy and product development.

Roberto Acosta graduated in 1996 from the University of New Hampshire (UNH) with a Bachelor of Science in Electrical Engineering. This thesis work completes the requirements for his Masters of Science in Engineering and Management from the Systems Design and Management Program at the Massachusetts Institute of Technology.

Acknowledgements

First I would like to thank Dr. Eric Von Hippel for his insight, guidance and many lessons taught beyond the scope of this thesis. Lessons which have already helped to successfully make new and improved products.

In addition to Professor Von Hippel, I also received great support from many members of the Open Source Hardware community and I would like to thank all of those who helped me with this project.

MIT was a great experience. Many members of my cohort helped me along the way and contributed in different ways to make my time at MIT into a wonderful learning experience. I would like to thank Gadi Oren, Aparna Chennapragada, Jeffrey Manning, Jess Kopczynski and Luis Maseda.

Finally, I am grateful for the continuous love and support of my family; my wife Karina Liendo, and my parents Roberto Acosta and Virginia Oakes.

Cambridge, Massachusetts **Roberto Acosta**

May 9, 2009

Table of Contents

PART I Background 8

PART II Open Source Hardware 11

Definition 11

History 11

PART III Challenges with Open Source Hardware Products 14

PART IV Open Source Hardware and the Value Chain 15

PART V Evaluating Open Source Hardware Innovation and its potential effect in the Value Chain 17

Part VI Open Source Hardware at the Product Level 25

Open Source Hardware at the Product Level: Medical Instrument 27

Open Source Hardware at the Product Level: Cell Phone 28

Open Source Hardware at the Product Level: Free Telephony Project IPO4 30

Part VII Open Source Hardware at the Sub-System Level 31

Part VIII Open Source Hardware at the Core Level 34

Hardware Description Languages 35

Field Programmable Gate Arrays 37

Open Source Hardware at the Core level: OpenCores.org 38

Part VIII Case study of designing an Ethernet framer through available Open Source Hardware Cores vs User Developed Cores 40

Part X Open Source Hardware & Business Models 43

Part IX Conclusions 46

Appendix A: Design Files for Ethernet FPGA 49

Bibliography 82

Figure 1: Value Add Activities 15
Figure 2: OpenMedic ECG PC Board layout. Only partial information is available to reproduce the design. Source http://OpenMedic.org 28
Figure 3: Distribution and Number of visitors to OpenCores........ 39
Figure 4 : DSM of Implemented Design........ 41

PART I Background

I began the System Design and Management (SDM) program at the Massachusetts Institute of Technology in 2006. During the first year I attended a course titled "Innovation in the Marketplace". The course taught amongst other valuable lessons the idea of distributed innovation. I began my professional career in hardware development in an aero/defense company. After five years of hardware development I decided to take part in the telecommunications boom of the late 1990's and joined a small networking startup called Avici Systems, Inc. One of the first things that I noticed was the stark contrast between the Open versus Closed Innovation[1] paradigms between the two sectors. While Sanders, A Lockheed Martin Company, my former employer relied heavily on internal ideas and had very low labor mobility; Avici constantly sought external partnerships with other companies and academia. Avici was able to incorporate ideas from a PhD thesis on massively parallel processor architectures towards developing one of the fastest Internet routers at the time. After the Internet boom I like several other engineers returned to the aero/defense sector and found it had drastically changed. Labor mobility and the increasing capability of external suppliers had served as erosion factors that were contributing to the breaking of the large vertical silos within the aero/defense sector. In the early 2000 the return of many former and new employees to this aero/defense sector also helped to change the notion of "not invented here". Whereas before the aero/defense sector was reluctant to adopt strategies employed by small commercial companies, with the influx of new and returning employees that had worked in small startup environment old paradigms changed to the adoption of outside processes and technologies. Although at my current work environment there has not been a wide embrace of the outside innovation there have been some significant changes that have occurred in the last couple of years. First there has been an increased reliance on external suppliers to provide technology that was once developed internally. And secondly there has been a reversal in the policy of utilizing and participating in Open Source Software.

[1] Closed Innovation paradigms are understood as those in which a company looks inward for technical advances and relies on its own resources to develop most of its technical developments.

The change in approaches came mostly due to economic and schedule pressures brought on by the competition and the customer. On the one hand the customer was reluctant to fund development of software that was currently available for free on the Internet and was more interested in the company solving the "hard" or "challenging" problems. Development of software for commonly available drivers such as USB and Ethernet were of no interest to the customer. The customer was mostly interested in the development of core technology to help him maintain a technological advantage. Second, the schedule pressures placed on product development made it impractical to spend internal resources on the development of technologies that were easily available for free or could easily be outsourced for less to another company. Whereas before the notion of outsourcing or purchasing technology from outside the company would have been a difficult sell it was now recognized that in order to compete effectively and to deliver value to the customer this was the only approach.

Along the same lines of increased "openness" for the first time my employer began authorizing the use of open source software, with certain controls and restrictions. This was a radical departure from previous longstanding company policy to explicitly forbid the use of open source software.   At AVICI the utilization of "open source software" for development of protocol layer software such as MPLS or VLAN was widely used in order to rapidly develop and deploy new software upgrades to existing hardware on the field. At the time many companies were developing the same type of data carrying mechanism and had recently been handed to the Internet Engineering Task Force (IETF) for open standardization by another of the major telecom carrier developers, and it made sense to take available source code and modify it in order to work on the company's own router.

As of today open source software has made large strides in the industry, changing the way many value chains operate and businesses compete. Even at my current employer there are the beginnings of noticeable positive impact with the limited use of open source software in terms of competition and lowered development costs.

Although Open Source Hardware has not seen the same growth as some of the Open Source Software projects, recent changes in technology and design methodology have made it easier to implement this model. The goal of this thesis is to examine open source hardware, its potential impact to existing value chains and its current utilization in the industry.

PART II Open Source Hardware

Definition

There are many definitions of what constitutes open source hardware. The general consensus is that Open Source hardware is electronic hardware design that is "freely available under one of the legally binding recognized open source licenses". *The open source hardware includes schematics, diagrams and design rules that can be used, studied and modified without restriction and can be copied and redistributed in modified or unmodified form either without restriction, or with minimal restrictions only to ensure that further recipients can do the same.*[2]

History

Open source hardware has a long history and has gone through several cycles of growth. It can be argued that the first notion of open source hardware was in the early 1970's with groups such as the Homebrew Computing Club. In these early days of computer engineering a small group of engineers collaborated and openly shared designs of what would become one of the first personal computers. According to Apple cofounder Stephen Wozniak, "a lot of tech-type people would gather and trade integrated circuits back and forth"[3]. There was no official or formal organization and the main role of the club was to trade circuit designs amongst members. The theme of the club was "Give help to others" and membership to the group was mostly as a hobby. Wozniak states that schematics and designs for Apple I products were "passed around freely" and help was provided in order to help other members build their own systems. Free revealing in this manner, provided the designers of the Apple computer with early feedback on their initial prototype by other lead users and at least part of the motivation for free revealing was the increased reputation gained amongst peers. Open source hardware designs in this case relied on the ability of users to tie different components together. The use of breadboards made it such that users did not need manufacturers in the process of developing products. Information was

[2] Definition was derived from literature and sites dedicated to open source hardware.

[3] Wozniak, Stephen "Homebrew and How the Apple Came to Be" in Steve Ditlea, ed., *Digital Deli*, 1984

transferred from Innovating lead users to all users in the community through newsletters and informal gatherings.

From the early days of open source hardware at the "product" level, soon lead developers turned their attention into how to create the basic building blocks of the products. While the initial wave of Open Source Hardware was related to how to put the basic building blocks together the late 1970's to mid 1980's saw large development at the function level of the value chain. In the 1970's Carve Meade and Lynn Conway developed a method by which users could begin to design large-scale integrated circuits. Several designs were grouped into a single production run in order to reduce development costs. Designs were transmitted using the Arpanet to manufacturing sites in order to be built. This process allowed universities access to low cost production of highly complex integrated circuits but users outside of this system could not easily develop or replicate the same products[4]. Although there were several free development tools, the computing power needed to produce large designs was extremely expensive. In this phase of open source hardware users began requiring the services of manufacturers in order to develop their products.

The 1990's saw increased development of proprietary design tools and this increased the knowledge needed to design o a particular platform making it difficult to switch and <u>share</u> designs at the user level. Although the design tools allowed for easy transfer of design output to manufacturers the increased segmentation in the tool market made it difficult to share designs between different users. Board design tools did not easily translate from one CAD manufacturer to another, for example design rules and databases for a board being designed with a tool such as "Mentor Graphics" were not easily translated to that begin designed under the "Cadence" environment. On the IC design side design input was begin driven by the IC manufacturers making it difficult to translate designs from one type of IC to other types of IC. This made it such that one ended up designing for a particular "targeted device" on a particular "tool flow", making it difficult to reuse the design or share it with others that were not operating on the same platform.

[4] Smith, Gina "Unsung Innovators: Lynn Conway and Carver Mead" *Computerworld 2007*

This approach led to the creation of whole groups of digital designers segmented by manufacturer and/or design tool.

Today changes in tool design methodology and standardization of interfaces have made it easier for open source to thrive and for user innovation communities to develop. Open Source hardware is thriving in cases in which users require manufacturers and cases in which users require little to no manufacturing involvement. As will be shown later most of the growth appears to be in innovation communities in which manufacturer involvement is little and where users can easily share, modify and upgrade the hardware with low cost approaches.

PART III Challenges with Open Source Hardware Products

Open source hardware has several characteristics that make it challenging to succeed when compared to other successful models based on Open Source. Unlike Open Source Software products such as Apache, open source hardware products will always have some form of manufacturer involvement either in the purchasing of the building blocks to build a product or in building of the product itself. The fact that it is a physical design adds a higher cost to the initial design and to recursive changes in the trial and error cycle of product development. Communicating and documenting changes and improvements are not inherent in the design as it is in software-based products. A fix or upgrade to a version of a product based on Open Source Software can easily be delivered in an automated way with current communication systems. The fix or upgrade will natively contain the changes made to the source code. This is not the case with a new release of hardware, as it will typically require users to physically modify their product, a task that they may not be able to perform. Developing hardware products can encompass a large range of expertise, from antenna design to digital logic and printed circuit board design. Some of this information is difficult to encode in such a manner that it is useful for others to reproduce, modify and recreate. Manufacturing techniques also present their own challenges as the more difficult a design is to build the more a designer must rely in what a manufacturer can develop or spend considerable resources searching or developing alternative manufacturing technologies. The latency on hardware development cycles also affects the pace at which innovation can occur drawing users and developers to established platforms and away from potential new designs that are not as stable or that do not have a history of continuous improvement. Hardware development cycles can range from a few weeks to several months[5] making it difficult to keep up with existing production cycles for products created by established companies with closed designs. This can affect open source hardware products since these types of products will need to capture and establish their own "ecosystem" of software developers and users in order to succeed.

[5] Typical Hardware development time for a PCB is 1 week for fabrication and one week for assembly. ASIC design of masks is in the order of a few months.

PART IV Open Source Hardware and the Value Chain

Open Source hardware today exists at most stages of the value chain, from low-level design gates to functional cores that can perform tasks such as image analysis to platforms and complete system/product solutions. Figure 1 shows the value chain or hardware from Atoms to Solutions and an Open Source Hardware example that exists today at each stage.

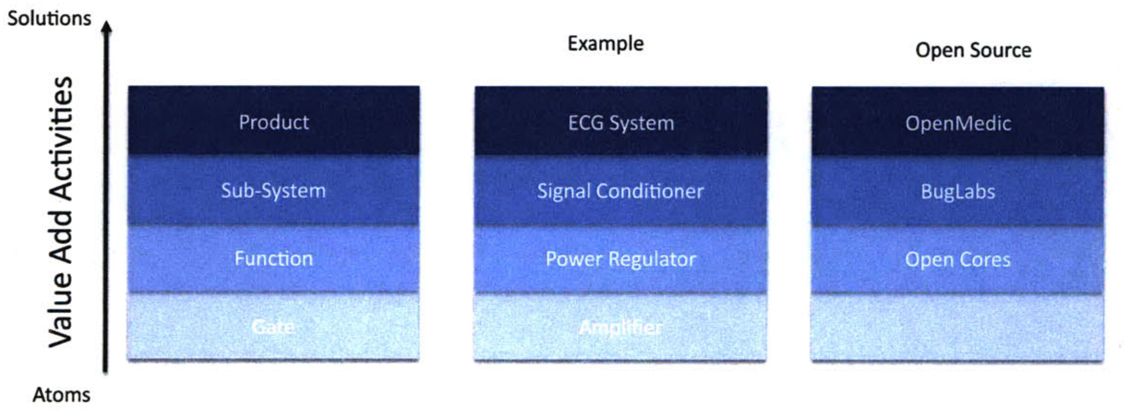


Figure 1: Value Add Activities

At the bottom of the value chain "atoms" such as transistor level gates exist. Production of gates at this level can be extremely capital intensive, and traditionally one does not build individual gates but at the most basic level a few low level functions such as small drivers, buffers, amplifiers and other basic building blocks. Current state of the art development of integrated circuits in a 45nm process can cost upwards of \$3 million dollars, with older technology 17um technology costing anywhere between \$150k-300k[6]. It is clear that one can develop Open Source hardware at this level but activity is typically limited to sharing the physics and material implementation of the devices. Development of hardware at this level can be extremely costly in terms of fabrication because it relies heavily on economies of scale. Companies such as IBM tend to develop a process and open source the methodology so that other corporations can utilize their fabrication facilities. This method helps owners of semiconductor facilities continue to develop innovative technologies while at the same time reducing the cost of owning and

6

maintaining the fabrication facilities. By open sourcing the technology IBM draws developers towards its fabrication process and helps to maintain production levels high while at the same time generating enough revenue to continue to develop processes that are innovative. There is no "open source of the design" but mostly of the process.

One step above the gate level, stand-alone circuit designs are available in the form of electronic schematics or cores. These designs typically relay on hardware only and require little to no external knowledge of system operation. Examples of designs at this level can range from simple controller such as universal serial bus (USB) drivers, Ethernet framers and more complicated functions such as CPU's and video chips. The cores by themselves provide valuable functions but need to be tied to other cores in order to perform a partial solution for the user. The integration of functions to form larger subsystems requires the user to understand both the core implementation and the interface characteristics. Developers of open source hardware typically generate custom cores through less expensive manufacturing paths such a reconfigurable logic or by wiring discrete atoms together. Developers have been known to generate highly integrated cores but this approach usually involves a higher cost and greater interaction with manufacturers.

An example of a subsystem is a Video Processing core tied to a display driver and an associated LCD screen in order to form a touch screen interface. This type of display/interface subsystem can be found in newer smart phones. This type of subsystem allows for the user hardware to display images and to receive input from a user but does not provide a full solution to a user need, it can however be used by a developer as a part of a system for a phone or a gaming device.

On the product side there are cases in which the entire product is truly open including schematics and software to re-engineer the entire product and other cases where only a small number of interfaces to the product are available. In some cases only portions of the hardware are "open" these cases, which typically refers to cases in which all documentation is provided to make the hardware function but no details as to how the hardware is built[7].

[7] This has been referred to as Open Hardware.

PART V Evaluating Open Source Hardware Innovation and its potential effect in the Value Chain

Von Hippel demonstrated that there was a continuing trend towards *democratizing innovation*[8]. By democratizing innovation Von Hippel meant that users were increasingly capable of developing products and solutions by themselves. Von Hippel also explained the process by which this shift was occurring and how innovation by users complemented the innovation done by manufacturers. Open Source Hardware can help drive this innovation by functioning as a source of designs which users can quickly leverage to build upon and create new products. Von Hippel developed a series of attributes to see where and how innovation was being democratized. These attributes taken from his book *Democratizing Innovation*[9] will be used to look through the value chain and see if the new sources of innovation are relying on open source hardware. The attributes will help to determine in a qualitative way if and how open source hardware is being utilized to build new and innovative products. In particular the following attributes of widespread innovation qualified by the use/or role of open source hardware will be utilized:

-Evidence of open source hardware developed or utilized by lead users.

-Need for custom solutions at several levels of the value chain and whether those custom solutions are based on Open Source Hardware.

-If the cost of implementation and use of open source hardware at that stage can reduce the cost of building hardware or if it is cheaper to purchase existing hardware outright.

-User low cost innovation niches along with information stickiness at each level.

---

[8] Von Hippel, Eric Democratizing Innovation. Cambridge, MA MIT Press 2005

[9] Von Hippel, Eric p 31-40

-Evidence of free revealing and reuse of open source hardware not only the of the source itself but of the manufacturing and development process.

- Existence and participation in innovation communities that develop open source hardware.

-Availability of toolkits in that are assist development with open source hardware.

Activity or evidence of innovation based on open source hardware present at any stage of the value chain can point towards a strong possibility that open source hardware can begin to displace or change established revenue models and organizations. In some cases manufacturers will have to retool to accommodate more of the activity being done by users as Von Hippel established[10]. In other cases firms may see their business models in direct competition with open source hardware.

Von Hippel observed the following characteristics of lead users "Lead users are at the leading edge of an important market trend and are currently experiencing needs that will later be experienced by many users in that market"[11]. And secondly they "anticipate relatively high benefits from obtaining a solution to their needs and so many innovate"[12]. Clearly if lead users have adopted Open Source Hardware as a vehicle to develop products or subsystems it is a an indication that Open Source Hardware can play an important role in future innovations and be the basis for future products in the market, if lead users continue to develop products and hardware designs with little use of open source hardware or based on current closed systems then it is likely that the current players will see their positions unchallenged. Also the utilization of Open Source Hardware by Lead Users can be used as an indication that Open Source Hardware can provide a solution space not available through current existing models.

[10] Von Hippel (2005) pg

[11] Von Hippel (2005) pg 22

[12] Von Hippel, Eric (1986) Lead Users: A source of Novel Product Concepts. Management Science. Vol 32, No 7, July 1986

Von Hippel also explained "*high rates of user innovation suggest that users may want custom products*"[13]. Von Hippel also explained that if individual users or firms want something different in a product type, it is said that heterogeneity of user need for that product type is high[14]. Open source hardware will be analyze in this context to see if it can be useful in providing a better custom solutions in some particular markets than those provided by hardware commonly available. For example can a user generate custom solution based on Open Source Hardware better than that available by an established company? Will users/developers be willing to spend resources and time working on a custom solution to obtain a custom solution at a particular point in the value chain and will they be willing to open source the design?

By having access to the hardware documentation schematics and diagrams users can gain insight into a particular subsystem or product and modify its characteristics to better suit their needs. Apache web server is an example of an open source software product whose specific characteristics and configuration allowed users to modify and develop web server security not available through other software systems, as described by Von Hippel[15]. Many adopted Apache since they could modify it better to suit their needs. The result of this activity has pushed the open source based Apache server as the preferred vehicle for developing web server software. The question remains if the same can be said for an Open Source Hardware system. Some trends such as the *Open hardware Foundation* were formed to meet *the efforts of the Open Graphics Project* whose aim is to provide amongst one of its charters the creation of a Open Source Graphics chip set which would indicate that there are enough needs unmet by current Graphic Chip sets that would make users want to generate their own graphics integrated circuit architecture and develop their own boards and circuits. This stands in stark contrast to other very active groups such as the General Purpose computation on Graphics Processing Units. This group was formed to develop software and applications based on Graphic processing units, which tend to achieve processing speed gains of orders of magnitude in computation versus available approaches. This group is formed of lead users that having experienced unmet needs in terms of the computational speed and processing provided by standard CPUs are looking for new

[13] Von Hippel (2005) pg 33

[14] Von Hippel (2005) pg 33

[15] Von Hippel (2005) pg 39

innovative solutions through the use of graphic processing units. The software developers have remained tied to established graphic chip sets such as the NVIDIA CUDA. This is one example of where an established graphic chip sets provide a solution not seen by standard CPUs, however there is only a shift in where the users will obtain their hardware they will trade standard CPU's for Graphic Chip sets but they will not invest heavily on new platforms. In summary the users will shift to purchasing a different type of processing IC and innovate by utilizing it in a different way but do not feel the need to seek out an open source hardware platform.

Information Stickiness and users low cost innovation niches will be looked for as potential factors affecting the use of Open Source Hardware. Von Hippel explained that the term of information stickiness is a measure of how costly is it to transfer information from place to place[16]. The stickiness of information is defined as the incremental expenditure required in transferring a unit of information to a specific location in a form usable by a specific seeker[17]. If the expenditure is low information stickiness is low, when the expenditure is high stickiness is high. Information stickiness can have a large impact on the use of Open Source Hardware. The cost required to transfer information in Open Source Hardware can vary greatly. In cases where schematics, integrated circuits and software is needed to recreate an Open Source Hardware solution the user must be able to access not only the design information but also the manufacturing requirements. There are cases where schematics are not enough to recreate a circuit board and a user must have access to board information such as which signals should be built with tighter constraints and which cases can have relaxed constraints. As an example a high speed Gigabit interface running at 1000 mb/second is more challenging in terms manufacturing than an I2C interface running at 100 kb/sec. Higher digital speeds require greater control in the manufacturing process than those running at slower speeds. The designer must be able to effectively communicate to the manufacturer the constraints required for a particular design. Although some newer CAD tools support encoding this type of information there are no set rules and tradeoffs in terms of the time and cost of manufacturing and the design requirements are usually made. A designer would have to measure the final implementation and write up the final

[16] Von Hippel, Eric (1994). " Sticky Information and the locus of problem solving: Implications for innovation." Management Science, 40(4) 429-439

[17] Von Hippel, Eric (2005) pg 67

constraints in order to fully communicate how a design was achieved, the designer may not have this information available since it may rest with the manufacturer or may find that there is an extra cost involved in order to capture this information. This can impact the utility of Open Source Hardware designs since they may not contain all of the information required to recreate a design. Von Hippel explained that low cost innovation niches as areas that are developed as a consequence of information stickiness and that information stickiness yields to information asymmetries that cannot be erased easily or cheaply[18]. Since hardware relays on manufacturers, information stickiness can be high in certain areas of the value chain. If that is the case then it is likely that Open Source Hardware will only impact the design aspect of the product development cycle, the building aspect will remain with the manufacturer. However if the information stickiness is low in order to manufacture certain types of hardware then it is possible that Open Source Hardware can greatly affect the manufacturing points in the cycles of product development, in these cases the manufacturing will loose some of the revenue that it can generate by providing services associated with building of the hardware.

Evidence of free revealing and reuse of open source hardware in terms of not only the of the design itself but of the whole system integration and development process. Von Hippel characterized "free revealing" proprietary information as meaning that the innovator voluntarily gives up all existing and potential intellectual property rights to that information and that all interested parties are given access to it[19]. Open Source Hardware is based on free revealing of the source code and schematics but it is bound by licensing. The type of licensing can have an impact on whether Open Source Hardware can challenge existing value chains. Free revealing on the manufacturing process will also be used as characteristic of possible success of Open Source Hardware. If evidence is available that such information is available it can be used as measure of Open Source Hardware activity that involves not only copying the initial design but that open source hardware is also gaining ground on the manufacturing aspect. Re-use is a good measure of value being generated by Open Source initiatives. If there is evidence that other designers are re-using the information that has been revealed it is an indication that other users

[18] Von Hippel (2005) pg 70

[19] Von Hippel (2005) pg 78

are benefitting from the information that has been made available. Von Hippel emphasized that valuable forms of re-use can range from those gaining general ideas of development paths to pursue or avoid to the adoption of specific designs[20]. There may be certain cases in which Open Source Hardware architectures are available but designers do not utilize them. If there is little reuse it can be an indication that although the information is available there in no interest or that there is not enough information available to make use of the desing in these cases it can mean that open source hardware designs will not compete with closed source designs.

Von Hippel defined innovation communities as nodes consisting of individuals or firms interconnected by information transfer links, which may involve face-to face, electronic, or other communication[21]. Von Hippel also explained that innovation communities are often specialized around certain technologies and serve as collection points for information related to narrow categories of innovation[22]. Another important value of innovation communities is that they can offer support in the form of tools and evaluation to developers by a large number of users. Innovation communities based around Open Source Hardware would be an indication that a large community of users who have embraced Open Source Hardware and are actively developing it. These communities are important because they can provide tool development and user-to-user assistance in the form of evaluation and hardware debug. This could be an indication of how widely diffused open source hardware is and how much momentum it has in certain areas of the value chain. It would also be an indication that traditional hardware development methods in which a user depends on established design houses or manufacturers for information on circuits could be changing. User to user assistance could mean that developers of certain types of hardware will see their positions challenged as users of the innovation community may increasingly rely on the knowledge of the of the innovation community rather than paying for design services from established companies. Another way that innovation communities can begin to challenge established companies is by directly developing hardware that competes with that of an established company. For example if a company is proficient at developing certain types of designs interfaces such as high-speed USB interfaces and an Open

[20] Von Hippel (2005) pg 88

[21] Von Hippel (2005) pg 96

[22] Von Hippel (2005) pg 97

Source Hardware innovation community begins to develop a competing implementation then the company could see some of its revenue disrupted

Von Hippel explained tookits as integrated sets of product design, prototyping and design testing tools[23]. The main goal of a toolkit is to enable non-specialist users to design high-quality, producible custom products. Von Hippel established the main characteristics of a high quality toolkit had the following attributes[24]:

(1) it will enable users to carry out complete cycles of trial-and error learning

(2) It will offer users a solution space that encompasses the designs they want to create.

(3) It will be user friendly in the sense of being operable with little specialized training

(4) It will contain libraries of commonly used modules that users can incorporate into custom designs

(5) It will ensure that custom products and services designed by users will be producible on manufacturers production equipment without modification by the manufacturer.

Availability of high quality toolkits for open source hardware will lead higher utilization and adoption of open source hardware by designers. If designs are available for users to leverage but the users do not have a way of manufacturing, testing or building variants of the designs then access to a particular hardware design may not be useful at all. However if toolkits are available for users to develop and build and experiment with open source hardware then we could see the rapid adoption and development of open source hardware. Traditional design houses could see some of their work be replaced by open source hardware developers since most of their activity would now be shifted to supporting the manufacturing of the hardware but not the design of it.

[23] Von Hippel, Eric (2002) "Shifting Innovation to Users via Toolkits", Management Science Vol 48, No 7, July 2002

[24] Von Hippel, Eric (2005) pg 154

Design houses traditionally provide assistance in design, layout, routing and material construction of PCB boards. If toolkits are available such that a user can design within certain layout and routing constraints then a design house may not be able to charge as much money for the services they provide. Another example would be the appearance of hardware platforms that allow the user to change/modify only certain portions. For example if a user wanted to test a variant of a hardware implementation of an algorithm the user may need to supporting circuits and infrastructure in order to properly test his/her design. If a collection of libraries that exist but are not open source the developer may choose to utilize this path since access to the existing circuits and infrastructure will greatly reduce his development time. However if a large collection of supporting libraries of open source hardware is available for the user to build upon the designer may choose this development path.

Part VI Open Source Hardware at the Product Level

Open Source Hardware at the product level is a more challenging to develop since expertise are required from multiple areas of hardware development such as: power supply design, analog systems and digital systems. Also at this stage one must have a system approach and understanding of all of the underlying and interlocking pieces. Other challenges of open source hardware at this stage are the need to have software capabilities and applications written for the product, even to get a crude prototype, which other users can then utilize. When developing open source hardware at the product level one must make provisions for either utilizing available software or developing the corresponding software. Successful integration with mechanical design and human factors is important in order to have a competing product that could possibly displace existing products in the market. The large integration of several components and subsystems increases the information stickiness, making it more difficult to extract all of the information in order to make it available for other users. The primary focus of open source hardware at the product level appears to be on competing with already existing solutions or platforms in cost. Developers and designers must be able to build and compete with existing market solutions and offer equal or better value to existing alternatives. The challenge with this approach is that currently established products have already been through several cycles of development and user testing making them more reliable. Also many of these established products already have supporting eco-systems around them that can be extremely challenging to replicate in a short amount of time.

Research found many currently available products based on open source hardware. Three of the products will be analyzed but the following generalizations apply to the few that were examined:

Developed products seem to focus on existing solutions and attempted to compete on cost or on minor approaches to satisfying a user need.

The cost of building a custom solution based on open source hardware compared to purchasing an existing solution varied quite a bit. In the case of highly available products such as cell phones the cost of building an open source solution was higher than purchasing a closed system. In cases such as a medical instrument or telephony router the cost was equal or less to those available from closed systems. One must note that the medical instrument was not licensed for medical use so it could not in theory be used as intended, if one were to factor in the cost of obtaining approval it could be conceivable that it would be just as costly for a user to choose the open source solution rather that to purchase an existing solution. The telephony system was considerably less cost those solutions available on the market.

Information stickiness was extremely high. Many of the products available as open source hardware were missing details on how to build them. For example in some cases board layouts and designs were available but there was not indication as to the copper layers used in the process the copper density used for the PCB layers, the materials utilized in the design or recommended manufacturing processes. Some of the information seemed to be posted just to satisfy the "open" initiative of the product but with no clear indication of how the information could be used to replicate the system. Information needed to adequately build the tooling and injection molding to take the products into production was not available. In some cases the main thrust of the design initiative was that of developing new applications for the system rather than extending the existing hardware. Instructions for coding and modifying the existing software were more readily available.

Free revealing and re-use. The intent of free revealing of the design was apparent as most products were licensed through GPL licenses. There appeared to be little or no re-use of the system. In most cases only the initial version of the prototype was available but subsequent versions of the design did not appear to exist, indicating that although the a product was developed with open source hardware in mind there was less of a follow through in next generations of the product.

Innovation communities were present in several of the examined cases however the communities were mainly geared towards developing software add-ons to the product platform not focused on developing next generations of the hardware.

Although there were certain toolkits available none of the would qualify as *high quality*. Toolkits were often present to help develop the parts of the software but none to support development of the hardware. Repositories of hardware information were unavailable or partially restricted leading developers to depend on "leaked" information in order to properly understand all of the hardware.

The following three examples are analyzes in more detail in the following sections: the development of medical instruments for electrocardiograms, the open source smart phone platforms OpenMoko and the open source telephony product. The examples were chosen since they represent a good cross section of utilization and degree of success.

Open Source Hardware at the Product Level: Medical Instrument

The *OpenMedic electrocardiogram* was developed with the goal of manufacturing and developing inexpensive basic medical equipment for use in poorer countries. The equipment developed leveraged existing off the shelf PC technologies with low-cost hardware/software based products. The Openmedic electrocardiogram represents a user low cost solution to develop an alternative to more expensive established products. The schematics and designs are available however important information regarding chipsets used, manufacturing, design constraints and designs rules are missing. There is not enough information available to adequately reproduce the design also there does not appear to be a lot of activity from other developers and only an initial version of the design was available.

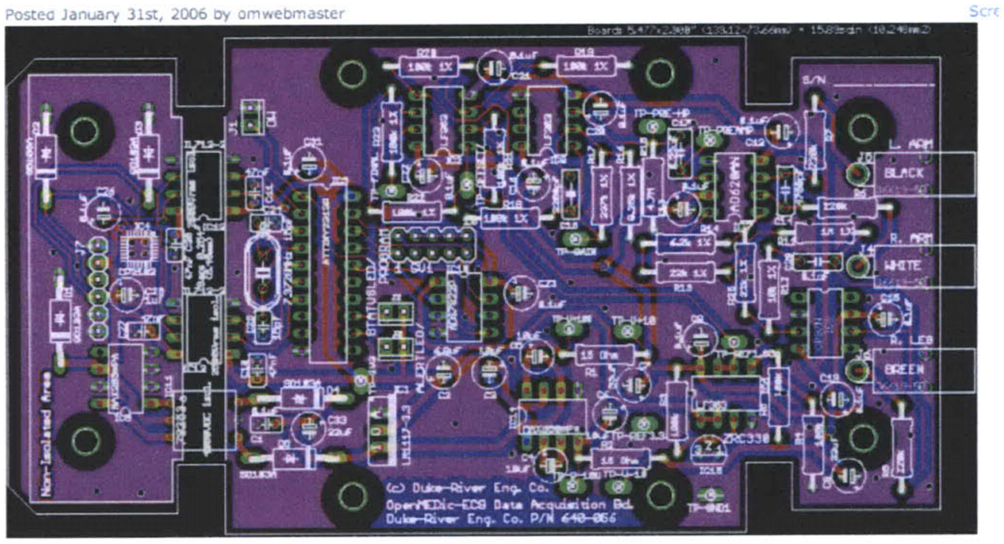

Posted January 31st, 2006 by omwebmaster
L. ARM
BLACK
R. ARM
WHITE
R. LEG
GREEN
Non-isolated Area
(c) Duke-River Eng. Co.
OpenMEDic-ECG Data Acquisition Bd.
Duke-River Eng. Co. P/N 640-056

Figure 2: OpenMedic ECG PC Board layout. Only partial information is available to reproduce the design. Source http://OpenMedic.org

Design reuse appears to be minimal meaning that existing developers of medical hardware as well as medical product designers may not see impact from an open source hardware based product such as this one.

Open Source Hardware at the Product Level: Cell Phone

Openmoko is aimed at building and delivering mobile phones with an open source software stack. The first iterations of the phone had a semi-open source hardware approach for developers and designers, as partial sets of schematics were made available for users to examine and reproduce. However the next generation schematics were not released due to NDA agreements with existing IC vendors and also certain law requirements that prevented full conformance with open source hardware although that was one of the developers original goals.

The Openmoko smartphone initially was perceived as an open source platform that could seriously challenge existing players in the market. The operating system and software stack was open making it easy for users and developers that wanted custom applications and products in the smartphone market to help develop their own. Manuals for most of the chipsets and instructions on how to disassemble and debug the hardware were made available along with a relatively inexpensive debug board ($99) that helped developers build new applications for the hardware. The product developers opened portions of the hardware design for users to review and upgrade. Also community forums were established so that users could help develop the next set of hardware requirements for future smart phone releases. For example missing features such as Wi-Fi connectivity on the first generation devices were added later as both hardware designers and software developers raised it as a major deficiency of the first generation. The initial launch of the OpenMoko suffered from serious delays on delivering stable working hardware: OpenMoko developers explained in an open letter to the community that "making changes to a product while in R&D stages can be quite painful[25]. But after all the incredible demand, post-November, we felt it had to be done. We had a string of bad luck that really hurt our productivity. Each hardware revision takes at least one month of time. Each month without stable hardware means serious delays of software"[26]. Even with several delays on the hardware development. Openmoko managed to sell 13,000 first and second-generation headsets.

However after three versions of Openmoko smartphone development the organization recently announce that while it would continue to develop the software platform aspect of the mobile communication device while it would stop developing and selling the hardware aspect of the platform. Steve Mosher responsible for marketing of the openmoko smartphone "admitted that the hardware design presented more difficult hurdles than anticipated. Too many design changes had led to delays. Openmoko had approached the hardware solution from the software development side, where things can simply be reprogrammed, which has since proved a fallacy". Although openmoko will continue to develop the software for opensource smartphones it remains to be seen if it will rely on manufacturers to develop the platforms on which its operating system will operate or if the organization will attempt a next generation phone in the future. The openmoko case exemplifies one of the challenges of hardware development; platforms tend to move quickly and both parts of the product must move at the same pace. If either the software or the hardware do not continue to improve the whole product can become obsolete. If this is the case an open source hardware design can loose most of its software

[25] Changes needed included Bluetooth connectivity which was shelved due to lack of internal resources

[26] http://lists.openmoko.org/pipermail/announce/2007-February/000003.html

support as developers may migrate to other more interesting or sustainable systems. Another lesson is that certain portions of the hardware can be closed at certain points, i.e. GSM encoders are bought of the shelf in order to minimize cost and therefore NDAs exist which prevent users form gaining access to the full hardware source. This approach leads to products not being fully open and users not being able to upgrade or modify certain portions of the hardware. Portions of hardware that have closed IP or IP protected by export controls will continue to present a problem for Open Source Hardware development as many encoders and communication subsystems tend to be closed and restricted due to legal and regulatory concerns.

Open Source Hardware at the Product Level: Free Telephony Project IPO4

David Rowe started the free telephony project with the main goal of developing low cost solutions in order to provide access to communication technology for all. The IP04 provides a low cost alternative to switch phone calls from analog lines to Internet through the use of Voice over Internet Protocol (VoIP)[27]. The IP04 system leverages the Asteriks open source telephony engine to deliver a full open source product that includes the software and the hardware. David Rowe utilized his knowledge in DSP to design a novel low cost telephony system based on Blackfin DSP processors. The Blackfin DSP processors are relatively low cost but provide enough computational capacity to provide all the necessary speech compression for multiple communication channels. Portions of the design itself are based on two previous open source hardware designs, which allowed the full system to be built and debug in a short amount of time. The product development is a good example of a team leveraging resources from around the globe. The hardware was developed in Australia portions of the software in Canada and the volume production is performed with a manufacturing in China. The result of the work is a telephony system that sells for 50% to 70% less than competitive products[28]. The product architecture also consumes less power than competing alternatives.

[27] Rowe, David "The IP04 Open telephony hardware for developing regions"

[28] Pika closed systems retails for USD1200 vs. IPO4 for USD450.

Part VII Open Source Hardware at the Sub-System Level

Open Source Hardware subsystems are building blocks of functions available for users in which to integrate modules to develop products or who want to rapidly construct prototypes of partial portions of a system. The building blocks are not a product itself but can be stitched together with other hardware modules or other additional sources to build different products or prototypes. The building blocks typically consist of highly integrated hardware functions plus some sort of software development platform for the user to customize. The hardware only provides a building block to which the user can add external capabilities. There is large lead user activity on this area from users developing from simple web based servo controllers to autopilots with stabilization and GPS navigation developed for use in autonomous aircraft. The platforms available at this level provide users with the ability to customize solutions to their liking although the solutions are not as polished in terms of presentation as those developed at the product level, they do include some that are extremely complex in terms of integration.
The development cost of hardware at this level is typically based on small blocks ranging from a few dollars[29] to larger more complex blocks ranging in the hundreds of dollars. Designers typically invest in one of these systems and modify it rather than building it from scratch although complete schematics, board-manufacturing instructions are available under the Creative Commons Attribution Share-Alike licenses and the microcontroller libraries are available under the LGPL license. Users typically build their own additional hardware to complement the basic building blocks. The information for developing systems from these basic building blocks is not as sticky as users typically post detailed instruction and videos showing how to wire the additional hardware. Also the level of integration is not as complex as when dealing with full products.

Re-use of circuits and designs tends to be high since many users are developing around the same architectures basic architectures. For example a Kalman filter used for stabilizing a two-wheeled robot was used to implement a stabilization circuit for an autonomous aerial vehicle. Users not only utilize the ideas for improving existing products but they also use existing ideas for new solution spaces. Another example is a remote controlled for an RC. Several different

[29] A basic ardurino block costs $18.95 USD while more complex bugbase subsytems run upwards of $200 USD

approaches based on the same board are available and many are referenced to the original posting of the first approach. Here users are taking advantage of the direct access to hardware source code, previous user solutions and relatively low material costs to design different solutions to problems. Problem solving is also not limited single point solutions, as users tend to iterate and develop newer and in some cases more efficient and novel approaches to solving the original problem. Innovation communities do exist and although they tend to be segmented across the solution interest that users are seeking (i.e. designers who want to develop servo-mechanical controllers will group together designers interested in navigation systems will group together) the basic building blocks tend to tie all the solutions back to the basic building blocks through web-based communities, ensuring that solutions developed in one space are available for other developers to utilize.

Custom solutions at this level tend to prevail at this level as users are developing specific solutions for their own interests. In some cases there are off the shelf commercial comparable products but users still tend to develop their own solutions due to better tailor the product for their own solution and in some cases for cost reasons. For example accelerometers and GPS blocks were tied together in order to increase the precision of the navigation system of a RC controlled aircraft. Off the shelf solutions that incorporate all of these features are available but are usually priced higher than those that users can develop and build on by themselves. A commercial RC GPS based system was found to be somewhere between 16000 USD[30], where as the cost of the open source hardware solution was around 200 USD[31].

The following cases based on open source building blocks were examined: the Arduino platform and the BugLab system. Arduino is an "open-source electronics prototyping platform based on flexible, easy to use hardware and software". The prototyping system revolves around a set of basic building blocks that perform specific functions such as a Bluetooth interface, an USB interface and a main processing board. The boards are all open source with schematics, diagrams and manufacturing instructions available for users to create build their own although most users tend to purchase built boards and then proceed to modify them. The community is organized around a central website and discussion site which links users from across the globe

[30] dragonfly X6 quoted price http://www.draganfly.com

[31] Ardupilot board http://diydrones.com

providing an efficient electronic transfer links of information. Specialized toolkits are available in the form of tutorials at every level to more complex "hacker" examples. A user can carry a trial and error learning of different modules characteristics by following the examples listed in order to get basic functionality down before attempting more complex hardware and software interactions. More experienced users are steered to a set of links with examples where other user generated designs are listed with the goal of having users extend and modify existing hardware capabilities. The language used to program the hardware is geared for non-software programmers but the compiler supports direct use of C++ for more experienced users. Libraries for commonly used functions and extensions of the hardware are available for users to leverage directly in order to speed hardware and prototype development by the users. Users are encouraged to share their ideas, projects and issues with the community in order to obtain different ideas or to gain direction on how a similar problem was solved. The main development site links users with other users or organizations and new users are encouraged to list themselves on the site in order to participate in the innovation process.

BugLabs is based on "open source functional hardware modules"[32]. Although hardware source is open and available for the modules themselves the main thrust of the group and the modules is in developing products by tying the modules together rather than modifying the physical modules themselves. The modules are highly integrated by function meaning that a building block may already contain one or more subsystems. The modules allow interconnectivity to a main "base" which is the core building block of any system. The ability to rapidly connect different blocks to perform different functions allows rapid prototyping and implementation of new user products with no need for manufacturing. The resulting prototypes have a more polished look and feel since the electronic modules are already mechanically encased. All of the information required to program the modules is available with supporting toolkits by the vendor such as system development kit for software to run the various modules, manuals for interconnectivity and internal schematics for users who want to gain learn or modify the modules.

[32] http://www.buglabs.net

Part VIII Open Source Hardware at the Core Level

Cores are series of circuits created to perform a particular function and can rage from relatively simple designs of 100+ logic gates to complete systems of >1M logic gates. The cores are typically of digital logic although; there exist some that can perform analog functions. The growth of Open Source Hardware design in this area has happened during the last 15 years and is due mostly to changes in hardware design paradigms, changes in technology and increased computing power.

Hardware implementation at this level can be done by wiring a large amount of discrete logic gates together or through the other preferred methods of building Application Specific Integrated Circuits (ASIC's) or by designing Field Programmable Logic Arrays (FPGA's) circuitry. The construction of logic designs through ASIC methodology requires more time investment but allows users to achieve higher level of integration. Typically firms who plan in building ASIC's involve multiple designers to create a single design. Designers need to be aware of the logic design, the physical technology (CMOS,MOSFET, etc) and have access to tools for the device construction. Design implementation involves a large number of steps, including schematic capture, layout, fabrication and packaging. The skills and knowledge required to develop hardware at this level ranges from electrical engineers for circuit design to mechanical engineers for packaging. This type of process also requires significant tool investment in specialized software, fabrication plants and can cost upwards of 1M USD per design. The design cycle for this type of circuit design is close to six-months and may require several iterations to finalize the design. Developing of cores for specific functions through the use of ASIC's can be an expensive proposition from which to develop hardware. Although in recent years ASIC vendors have developed high quality toolkits that enable users to design their own custom solutions at lower price points and with reduced production costs by making use of libraries which contain commonly available modules and which handle most of the physical implementation of the design allowing users to concentrate on the design itself. In the early 90's there was a convergence of factors that lead to alternative ways in which logic designs could be built. First design methodology transition from schematic design flow to the use of Hardware Description Languages (HDL's), second field programmable logic devices (FPGA) achieved densities comparable to those of small to medium sized Application Specific Integrated Circuits (ASIC) and finally design methodology was streamlined.

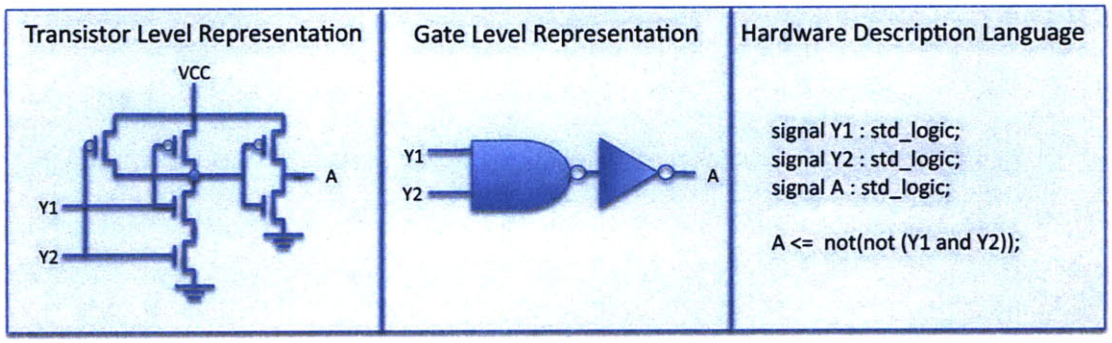
Hardware Description Languages

Hardware description languages are programming languages used to model the intended operation of a piece of hardware. The two most commonly utilized languages are VHDL and Verilog. VHDL appeared in 1980 when the USA Department of Defense (DOD) established the Very High Speed Integrated Circuit (VHSIC) program to create a standard hardware description language that was self-documenting and followed a common design methodology[33]. In 1987 the Institute of Electrical and Engineers (IEEE) ratified it as standard IEEE Standard 1076. Verilog had its origins in a CAE company called Gateway Design and after several iterations and modifications was reviewed and adopted by the IEEE as IEEE standard 1364. Of all the designs submitted to ASIC foundries in the 1993, 85% were designed and submitted using Verilog[34]. Today most ASIC designs and FPGA designs are done through Hardware description language design methodology. Hardware description languages help to decrease the information stickiness through the following characteristics: First they allow hardware design to be done through a standardized programming language. In this case a user that utilizes a hardware description language can easily transfer designs with no changes, cost (other than those needed to send the design) or extra steps required to convert the information from one form to another. Most new development tools can handle both of the standards in a mixed mode implementation reducing even the need to represent the entire design fully in one of the languages in its entirety. Logic designs can now be easily edited simulated and shared with other users without having to deal with tool compatibility. Second the increased level of abstraction of the code makes it easier to create and modify large amounts of hardware logic through code. Information transfer costs are reduced because although less of the design needs to be represented all of the needed information is available to recreate the design. Hardware languages add several layers of abstraction and low-level implementation details are generally not needed in order to understand

[33] Smith, Douglas J. HDL Chip Design Madison, AL Doon Publications 1996

[34] Smith, Douglas J. (1996) pg 24

the logic design implementation. Functionally the logic design behaves the same but the amount of information needed to understand it, replicated and implemented is greatly reduced. Lastly hardware design can now be technology and device independent, allowing for designs to be for the most part generated independently from the targeted device and easily portable. This allows users that implement a design to target different physical technology implementations without having to understand the physical structure of the device and its underlying implementation. Instead of having to understand the design as targeted for a particular physical implementation, with HDL's a design is first implemented in code and then targeted towards a particular physical implementation. Increased portability leads to increased amount of re-use a designer or user can develop an HDL library based on components that he or she may want to re-implement for different designs.


Transistor Level Representation
VCC
Y1
Y2
A
Gate Level Representation
Y1
Y2
A
Hardware Description Language

```
signal Y1 : std_logic;
signal Y2 : std_logic;
signal A : std_logic;

A <= not(not (Y1 and Y2));
```

Figure 3: Different Representations of the Same Circuit: The circuit represents a conjunction function and is first implemented via all required transistors; the second representation is utilizing ANSI/IEEE STD 91-1984 symbols and the last one in VHDL.

Figure 1 shows three different representations of the same digital logic design. The first circuit is a transistor level implementation of the design. This representation shows the actual transistor wiring as would be implemented in the physical integrated circuit. In order to properly model and construct this design a user must be aware of transistor behavioral characteristics as well as small and large signal modeling at the transistor level. Furthermore the user would need a tool capable of describing all of the individual nodes and transistor characteristics in order to test, implement and/or modify the circuit. Most of these tools incur some cost, although there is at least one Free and Open Source Tool available[35] to design and build circuits at this level the tool is not compatible with those available and established as industry standards.

[35] Magic 7

Gate level representation is one level of abstraction above transistor level implementation; it is easier to implement designs through this methodology than through direct transistor representation, however one must have access to tools and development suites that can read and generate schematics. Initially this type of design methodology was the preferred method for logic design, but schematic capture can be extremely time consuming and difficult to maintain. Schematics must be generated for a particular tool environment and do not easily transition from one design environment to another. Schematics can also be difficult to interpret and are more susceptible to mistakes when attempting to copy or to share.

The last representation is a Hardware Description Language representation of the circuit. Although logic design through this method has some its own challenges, the circuit is represented in a text format. The text is then fed into a synthesizer (the equivalent of a compiler for software) that maps the design into logic gates utilized by the design flow to implement the design. By working at this level of abstraction designs can be easily shared and made available to others. Users only need a freely available text editor in order to generate designs. Tools for synthesis and physical mapping for the device are available as part of open source initiatives and some IC vendors allow users free versions of their tools for designs that are less than a certain number of gates.

Field Programmable Gate Arrays

Ross Freeman and Bernard Vodershmitt first created Field Programmable Gate Arrays (FPGA's) were first created in 1985 while working for Xilinx. FPGA's are semiconductor devices that can be reconfigured by the customer after the IC has been manufactured. FPGA's have helped Open Source Hardware development by reducing the number of cycles in which the manufacturer is involved in the process. The ability to reprogram the FPGA provides a way of developing hardware that reduces the cost of experimentation and allow developers. Designers can iterate several times around a particular design without having to remanufacture a the digital logic. This approach diverges from the traditional ASIC path in which a fabrication house must be available in order to develop logic designs of high density. Although the unit cost of an FPGA is considerably higher than that of an ASIC when developed in volume the non-recurring engineering cost to produce a single FPGA is orders of magnitude lower than that of an ASIC. The logic density of an FPGA cannot achieve what is possible for an ASIC today however current FPGA sizes have grown from early 1200 equivalent logic gate size to 7 million equivalent gate sizes more than enough for large-scale integration. FPGA's can be considered a *low cost innovation niche* in which users that have heterogeneous needs and sticky information can develop their solutions without having to resort to manufacturer knowledge for design implementation. As long as logic designs are kept within the boundaries of the FPGA a user has the ability to design, build, run and analyze a design in house greatly decreasing the trial-and error cycle of product development. FPGAs also allow for users to experiment new designs without having to pay for manufacturing costs.

Open Source Hardware at the Core level: OpenCores.org

Core development has been on of the areas that have seen accelerated growth in terms of Open Source Hardware. Groups such as OpenCore.Org can boast membership of over 20,000 individuals and currently has over 400 designs developed and available for users to download. The membership of the group is distributed worldwide and ranges from skilled professionals to enthusiasts. Figure X was provided by opencores.org and it shows a regional breakdown of activity over the last month. The objective of the group is to "design and publish core designs under a license for hardware modeled on the Lesser General Public License (LGPL)". One of the main reasons for founding the group was to "reduce the excessive time-to-market and excessive cost in building deep submicron designs with millions of gates" with a technical solution that involved "the reuse of cores and the shared expending workload of verification". The Cores are typically coded in one of the standard hardware description languages and stored under revision control at a centralized deposit. OpenCores began with relatively simple functions such as CORDIC functions but is currently targeting development of more integrated and complex designs such as multicore processor solutions and high-end digital signal processing engines. It has adopted the Wishbone open source hardware computer bus to allow users to better integrate different cores and allowing it to move a bit up the value chain as users are beginning to build and develop systems on a chip. Part of the toolkits for users and developers is the ability to obtain a set of schematics to build their own test board through or to purchase a low cost development kit. The board serves as a physical interface in which users can further validate their designs. Statistics of individual core activity are logged as well as recent updates, bugs or feature requests. There is typically a group of people listed to serve as core or project maintainers providing some continuity and guarantee that the core will be properly supported. One of the important aspects of this community is that hardware vendors and commercial developers also support it. For example some cores are developed in Bluespec, which has higher level of abstraction than other hardware description language. Users who want to learn more or implement cores in this more innovative language will most likely be pointed towards tools and services provided by the Bluespec company itself. Users/companies of opencore.org not only utilize the area and to obtain new designs but also as a connection point to a broad network of users for services and designs that they provide.

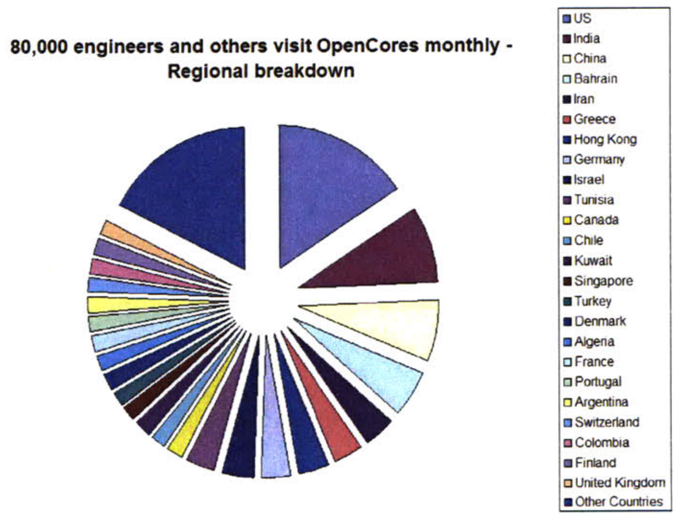


Figure 3: Distribution and Number of visitors to OpenCores

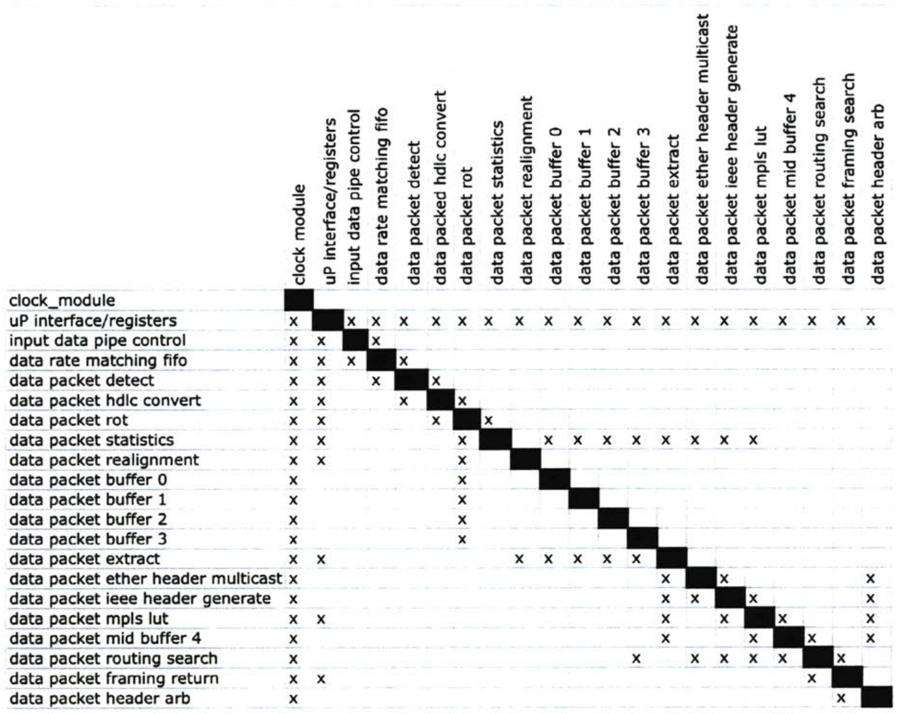
Source OpenCores.org

Part VIII Case study of designing an Ethernet framer through available Open Source Hardware Cores vs. User Developed Cores.

The following design was developed through full code over the last couple of months. Metrics for the design will be used to compare it those available through open source cores. Development of a core was chosen because the hardware could be modeled/simulated through the use of Hardware description languages without having to relay or depend on manufacturing of physical item.

The main goal was to obtain a series of metrics to compare an open source solution with that of a custom design. The designs will be compared for overhead, functionality, time spent on design cycle and verification as well as resource utilization on of the FPGA.

The design implements an Ethernet framer and includes the following main components: a data rate matching fifo, a High-Level Data Link Control block to properly detect opening and closing flags [7E hex] and properly detect and convert user data, an Ethernet framing block and FIFO interface for a physical interface layer. The design includes commonly used cores for Static Random Access Memory (SRAM) and uP interface. The following block diagram represents the implemented blocks. Highlighted blocks were replaced with commonly available blocks from OpenCore.org . The Design Structure Matrix represents the component interactions from the individual blocks.



	clock module	uP interface/registers	input data pipe control	data rate matching fifo	data packet detect	data packed hdlc convert	data packet rot	data packet statistics	data packet realignment	data packet buffer 0	data packet buffer 1	data packet buffer 2	data packet buffer 3	data packet extract	data packet ether header multicast	data packet ieee header generate	data packet mpls lut	data packet mid buffer 4	data packet routing search	data packet framing search	data packet header arb
clock_module	■																				
uP interface/registers	x	■	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
input data pipe control	x	x	■	x																	
data rate matching fifo	x	x	x	■	x																
data packet detect	x	x		x	■	x															
data packet hdlc convert	x	x			x	■	x														
data packet rot	x	x				x	■	x													
data packet statistics	x	x					x	■		x	x	x	x	x	x	x	x				
data packet realignment	x	x					x		■												
data packet buffer 0	x						x			■											
data packet buffer 1	x						x				■										
data packet buffer 2	x						x					■									
data packet buffer 3	x						x						■								
data packet extract	x	x							x	x	x	x	x	■							
data packet ether header multicast	x													x	■	x					x
data packet ieee header generate	x													x	x	■	x				x
data packet mpls lut	x	x												x		x	■	x			x
data packet mid buffer 4	x													x			x	■	x		x
data packet routing search	x												x		x	x	x	x	■	x	
data packet framing return	x	x																	x	■	
data packet header arb	x																			x	■

Figure 4 : DSM of Implemented Design

The design was taken through the synthesis aspect of the design cycle to obtain the utilization metrics represented below:

Metric	User Developed	OpenCore Developed
Logic Elements	33,769/81,264 (42%)	48,904/81,264(60%)
Combinatorial Functions	29,934/81,264 (33%)	38,307/81,264 (47%)
Dedicated logic Registers	21,388/81,264(26%)	29,934/81,264(37%)
Memory Bits	1,174,541/2,810,880(42%)	1,998,633/2,810,880(71%)
Verification	2 weeks	1 week
Design Time for Modules	4 weeks, 1 week integration	2 weeks to integrate

The design implemented with cores had higher utilization, this can be traced to the higher functionality not required by the design but included with the framing core. The design time was significantly reduced and although there was an added cost in learning of the details of the core this was considerably less than the time needed to engineer a custom solution. The cores available were fully verified resulting on only the interfaces between cores and user logic needing verification resources and time. The only penalty besides the building of capabilities by learning to implement the functions was a design that was not optimized for the individual solution.

Part X Open Source Hardware & Business Models

Henry Chesbrough explained some of the dynamics that business were facing when investing in innovation. First useful knowledge and technology are becoming increasingly widespread and distributed across companies of different sizes and many parts of the world. Second there are two distinct market dynamics that are contributing to the rising cost of innovation[36]. The first dynamic is that of an increasing cost of investment in order to develop new and advanced technology. Chesbrough showed that the cost of a new semiconductor fabrication plant had seen a 100% increase in of the cost building the plant over 20 years to reach a total figure of $3 billion[37]. A second market dynamic was that of shorter product life cycles. An example cited was that of the expected model of hard-drive retaining its leadership position in the market having shifted from two-three years down to six to nine months before newer and better solutions appeared. Chesbrough demonstrated that the combination of these two forces reduced a company's ability to earn a satisfactory return on its investment in innovation. If a company decided to open its business model it could attack the cost side problem by leveraging external R&D resources to save time and money in the innovation process.

A company that engages in Open Source Hardware development can leverage innovative solutions from outside the company in most cases for relatively little cost. A key idea from Open Innovation as stated by Chesbrough was the fact that "not all of the smart people work for you". A company that invests in open source hardware can have access to innovation communities developing novel and advanced products as well as access to lead users who are developing commercially attractive products[38]. Links to these communities allow businesses to better serve some of their customers by gaining insight into their current needs and applications.

[36] Chesbrough, Henry Open Business Models. Boston, MA Harvard Business Press (2006) pg 10

[37] Chesbrough, Henry pg 11

[38] Von Hippel, Eric (2005)

Businesses can position themselves in a manner such that they can profit from open source hardware. The following approaches described by Chesbrough for open source software companies have equivalent open source hardware approaches:

-Selling support, service or customization of hardware. A company that invests and participates in developing open source hardware can benefit from users seeking support or customization of open source hardware designs that they are utilizing. An example of this is the Arduino hardware whose founders along with making a small profit by the sale of their open source circuit. Gain most of their revenue from client who wants to build devices based on their board and who hire the founders of the company as consultants.[39]

-Versioning of Hardware, lower level IP blocks used as entry offerings with higher performance cores or value added offerings. Some core developers have favored this approach by providing certain lower level cores for free as a trial basis for user to test and then charge a premium higher performance closed designs or added features such as higher performance or lower power consumption.

-Integrate the Open Source hardware to build higher value products that a customer may not want to manufacture. Companies such as VIA have released a full set of schematics and CAD diagrams in order to build an open source notebook. With this information anybody can build a notebook through a low cost manufacturer. The advantage for the company is that the design is based on their microprocessor and associated IC's. By releasing the design for the full product design, the company will gain market share if consumers decide to build based on its desing and integrated circuits.

-Provide complements to the open source hardware by building proprietary modules or tools that work with existing open source hardware modules. Companies such as Bluespec have released open source designs to OpenCores. Developers who utilize this cores will see the

[39] Thompson, Clive "Build It. Share It. Profit. Can Open Source Hardware Work?" Wired Oct. 2008

benefit of designing with Bluespec modeling and desing solutions and will see the value of investing in one of several tools provided by the company.

Businesses also have strong reasons participating in open source hardware through the release of some of their designs. First old or unused designs may find new niches or applications by members of the open source hardware community. A company that maintains manufacturing capacity for certain parts of the design or certain portions of a design can see an increase in sales of an old design if an open source community finds new and novel approaches utilizing the components or technology that were not originally though of by the company. A company can also obtain revenue from support services for a design.

Part IX Conclusions

Open source hardware activity is highly concentrated in the mid section of the value add activities of the value chain although activity is present at the top of the value chain. Products and activity at the top of the chain had relatively less re-use or modification than open source hardware at lower levels of the value chain. Most designers/developers utilized the open source hardware as a platform on which to develop software and applications not as a way to modify and improve the existing hardware. Software/Application related groups were active for the open source hardware platforms but in some cases could easily transfer their applications to other platforms meaning that hardware developed at a product level can be at risk of keeping a software development group tied to its system, This can make it difficult for open source hardware platforms to compete with newer and better solutions that can appear based on closed systems since in many cases only an interface layer is needed to program the device allowing for easy migration to from platform to platform.

Information regarding the hardware was complete for software developers but incomplete for hardware designers in terms of design rules and manufacturing information. In order to manufacture hardware at this scale a developer needs a greater amount of information from different sources and must deal with factors such as reliability, sustainability and manufacturability.

The sub-system level saw an increase of open source hardware development with many users tying subsystems to build their own products and solutions. Most users/developers were involved in the product development cycle and also interested in improving products available by other users. User communities tended to form along the building of particular sets of products and solutions that would compete with more expensive solutions available in the market. Products generated from the subsystems tended to improve existing solutions in the market at a reduced cost. If manufacturers begin developing better toolkits products developed from open source subsystems could begin to challenge established system integrators.

Open Source hardware at the gate level is the most ubiquitous. Open source hardware designs available at this stage can begin to challenge established core developers. In interviews with hardware developing firms some have begun to replace established purchased cores with cores available from areas such as opencores.org. Established core developers must either develop cores that offer distinct advantages over freely available ones or must offer a higher level of integration in order to compete effectively. For example a memory interface designed by a core vendor must prove to be much better on memory access times or power consumption that one freely available. Establish core developers must offer integrated solutions such as offering memory interface solutions coupled with memory queuing systems or must compete with better support and service. A design comparison between one designed with open cores versus custom cores resulted in less time in verification for the design created from open source. Most cores obtained had greater functionality than required for a specific function and were adequately supported and debugged leading to a faster development cycle. Open Source development communities utilized their vast human resources to develop highly complex defect free core solutions in a short amount of time.

Open source hardware is available at all stages of the value chain and developers with system thinking and those that can manage complex system interaction can take advantage of available resources to bring faster and better solutions to the market. There appeared to be no threat of "free ridership", "cloning" or "me too" companies who only copied the design and decided to market it for their own profit. Common to other businesses approaches an open source hardware based business relies on service and quality of the manufactured product. An open source hardware based business also depends on a community of developers and users who help to create the product. Establishing relationships with manufacturers such that high quality hardware can be built through long production runs and creating a community of developers and designers takes time and effort and cannot recreated easily. By freely revealing the hardware source the businesses obtain the benefit of developers to test and enhance their product while at the same time they generated a value add that strengthened the position of their product. Branding was an important aspect of open source hardware. Companies such as Arduino allowed their design to be copied but retained the name of the platform, that way they can protect their design/platform from others who may not manufacture it with the same standards that they had.

Developers and designers of open source hardware not only gained from a much better design product by utilizing a community of developers but also saw their reputation increased and in cases such as the open telephony project and the Arduino platform the lead designers were able to obtain sources of revenue by providing services and consulting to contacts made through their product.

Appendix A: Design Files for Ethernet FPGA

```
////////////////////////////////////////////////////////////
// File  : sdm_proj09.v
// Author: Roberto Acosta
//
// Description:
// This module includes the Ethernet framer module plus the I/O Ring
//
// All ports with suffix '_l' are low asserted.
////////////////////////////////////////////////////////////
// Revision History:
//
// Date                     Author    Revision
// Sdmprj Mar 05th, 2009    R.A.      First Pass
//
//
////////////////////////////////////////////////////////////

module sdmproj            (
                          clk_MAC_i,
                          clk_SYS_i,
                          sdmprj_reset_l_i,
                          sdmprj_mb_add_data_io,
                          sdmprj_mb_reset_i,
                          sdmprj_mb_start_i,
                          sdmprj_mb_busy_o,
                          sdmprj_mb_select_i,
                          sdmprj_mb_clk_en_l_i,
                          sdmprj_mb_int_l_o,
                          sdmprj_txsel_l_o,
                          sdmprj_fps_txf_o,
                          sdmprj_sop_txf_o,
                          sdmprj_eop_txf_o,
                          sdmprj_vtg_o,
                          sdmprj_txasis_o,
                          sdmprj_flct_o,
                          sdmprj_flct_lat_o,
                          sdmprj_txrdy_0_o,
                          sdmprj_txrdy_1_o,
                          sdmprj_chn0_tdat_p_i,
                          sdmprj_chn0_tclk_155_i,
                          sdmprj_chn0_tclk_155_l_i,
                          sdmprj_chn0_clk_155_o,
                          sdmprj_chn0_clk_155_l_o,
                          sdmprj_chn0_ext_rd_o,
                          sdmprj_tx_si_o,
                          sdmprj_tx_ld_o,
                          sdmprj_tx_sen_l_o,
                          sdmprj_tx_rst_l_o,
                          sdmprj_tx0_data_o,
                          sdmprj_tx0_be_o,
                          sdmprj_tx0_wclk_o,
                          sdmprj_tx0_ren_o,
                          sdmprj_tx0_oe_l_o,
                          sdmprj_tx0_wen_o,
                          sdmprj_tx0_ffl_i,
                          sdmprj_tx0_efl_i,
                          sdmprj_tx0_pafl_i,
                          sdmprj_chn1_tdat_p_i,
                              sdmprj_chn1_tclk_155_i,
                              sdmprj_chn1_tclk_155_l_i,
                              sdmprj_chn1_clk_155_o,
                              sdmprj_chn1_clk_155_l_o,
                              sdmprj_chn1_ext_rd_o,
                              sdmprj_tx1_data_o,
                              sdmprj_tx1_be_o,
                              sdmprj_tx1_wclk_o,
                              sdmprj_tx1_ren_o,
                              sdmprj_tx1_oe_l_o,
                              sdmprj_tx1_wen_o,
                              sdmprj_tx1_ffl_i,
                              sdmprj_tx1_efl_i,
                              sdmprj_tx1_pafl_i,
                              sdmprj_vlanram_clk_o,
                              sdmprj_vlanram_cs_o,
                              sdmprj_vlanram_oe_l_o,
                              sdmprj_vlanram_we_l_o,
                              sdmprj_vlanram_adv_o,
                              sdmprj_vlanram_addr_o,
                              sdmprj_vlanram_data_io,
                              sdmprj_ram_clk_mirror_in_o,
                              sdmprj_statsram_clk_o,
                              sdmprj_statsram_cs_o,
                              sdmprj_statsram_oe_l_o,
                              sdmprj_statsram_we_l_o,
                              sdmprj_statsram_adv_o,
                              sdmprj_statsram_addr_o,
                              sdmprj_statsram_data_o,
                              sdmprj_ram_clk_mirror_out_i
                              );

/////////////////////////////////////////////
// SYSTEM
/////////////////////////////////////////////

input           clk_MAC_i;
input           clk_SYS_i;
input           sdmprj_reset_l_i;

/////////////////////////////////////////////
// uProcessor BUS
/////////////////////////////////////////////

inout [17:0]    sdmprj_mb_add_data_io;
input           sdmprj_mb_reset_i;
input           sdmprj_mb_start_i;
input           sdmprj_mb_select_i;
input           sdmprj_mb_clk_en_l_i;
output          sdmprj_mb_busy_o;
output[1:0]     sdmprj_mb_int_l_o;

/////////////////////////////////////////////
// Common Port Fifo Interface
/////////////////////////////////////////////

output          sdmprj_tx_si_o;
output          sdmprj_tx_ld_o;
output          sdmprj_tx_sen_l_o;
output          sdmprj_tx_rst_l_o;

/////////////////////////////////////////////
```

```
// Port 1 Interfaces
/////////////////////////////////////////////
// Channel 1 packets

input [15:0]        sdmprj_chn0_tdat_p_i;
input               sdmprj_chn0_tclk_155_i;
input               sdmprj_chn0_tclk_155_l_i;
output              sdmprj_chn0_clk_155_o;
output              sdmprj_chn0_clk_155_l_o;
output              sdmprj_chn0_ext_rd_o;

//FIFO

output[31:0]        sdmprj_tx0_data_o;
output[3:0]         sdmprj_tx0_be_o;
output              sdmprj_tx0_wclk_o;
output              sdmprj_tx0_ren_o;
output              sdmprj_tx0_oe_l_o;
output              sdmprj_tx0_wen_o;
input               sdmprj_tx0_ffl_i;
input               sdmprj_tx0_efl_i;
input               sdmprj_tx0_pafl_i;

/////////////////////////////////////////////
// Port 2 Interfaces
/////////////////////////////////////////////
// Channel 2 Interfaces

input [15:0]        sdmprj_chn1_tdat_p_i;
input               sdmprj_chn1_tclk_155_i;
input               sdmprj_chn1_tclk_155_l_i;
output              sdmprj_chn1_clk_155_o;
output              sdmprj_chn1_clk_155_l_o;
output              sdmprj_chn1_ext_rd_o;

//FIFO

output[31:0]        sdmprj_tx1_data_o;
output[3:0]         sdmprj_tx1_be_o;
output              sdmprj_tx1_wclk_o;
output              sdmprj_tx1_ren_o;
output              sdmprj_tx1_oe_l_o;
output              sdmprj_tx1_wen_o;
input               sdmprj_tx1_ffl_i;
input               sdmprj_tx1_efl_i;
input               sdmprj_tx1_pafl_i;

/////////////////////////////////////////////
// Memory Interface (L2 Table)
/////////////////////////////////////////////

output              sdmprj_vlanram_clk_o;
output              sdmprj_vlanram_cs_o;
output              sdmprj_vlanram_oe_l_o;
output              sdmprj_vlanram_we_l_o;
output              sdmprj_vlanram_adv_o;
output[18:0]        sdmprj_vlanram_addr_o;
inout [35:0]        sdmprj_vlanram_data_io;
input               sdmprj_ram_clk_mirror_in_i;

/////////////////////////////////////////////
// Memory Interface (Counters)
```

```
////////////////////////////////////////////

output          sdmprj_statsram_clk_o;
output          sdmprj_statsram_cs_o;
output          sdmprj_statsram_oe_l_o;
output          sdmprj_statsram_we_l_o;
output          sdmprj_statsram_adv_o;
output[18:0]    sdmprj_statsram_addr_o;
inout [35:0]    sdmprj_statsram_data_io;
output          sdmprj_ram_clk_mirror_out_o;

////////////////////////////////////////////
// MAC Interface
////////////////////////////////////////////

output          sdmprj_txsel_l_o;
output          sdmprj_fps_txf_o;
output          sdmprj_sop_txf_o;
output          sdmprj_eop_txf_o;
output          sdmprj_vtg_o;
output          sdmprj_txasis_o;
output[1:0]     sdmprj_flct_o;
output          sdmprj_flct_lat_o;
input           sdmprj_txrdy_0_i;
input           sdmprj_txrdy_1_i;




/****************************************\
*            Clock SYS                   *
\****************************************/

wire    clk_SYS_i;

IBUF_CLK        (.I(clk_SYS_i),.O(clk_MACdlli));

CLKDLL DLL_SYS          (
                .CLK0(clk_SYSinta),
                .CLK90(),
                .CLK180(),
                .CLK270(),
                .CLK2X(clk_2xSYSa),
                .CLKDV(),
                .LOCKED(),
                .CLKIN(clk_SYSint),
                .CLKFB(clk_SYS),
                .RST(~sdmprj_reset_l)
                );

BUFG  BUFG_SYS          (
                .I(clk_SYSinta),
                .O(clk_SYS)
                );

BUFG  BUFG_2xSYS(
                .I(clk_2xSYSa),
                .O(clk_2xSYS)
                );
```

```
wire                    sdmprj_tx1_wclk;
                        assign sdmprj_tx1_wclk = sdmprj_tx0_wclk;

/*****************************************\
*  CLOCK MAC & CLOCK CHNNET GEN.        *
\*****************************************/

CLKDLL FIFOTX (
                        .CLK0(sdmprj_tx0_wclk),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(),
                        .CLKDV(),
                        .LOCKED(),
                        .CLKIN(clk_SYS),
                        .CLKFB(sdmprj_tx1_wclk),
                        .RST(~sdmprj_reset_l)
                        );

/*****************************************\
*  CLOCK MAC & CLOCK CHNNET GEN.        *
\*****************************************/


wire        clk_MAC_i;
wire        clk_MACdlli;
wire        clk_MACdllo;
wire        clk_MAC;

wire        clk_2xMACdllo;
wire        clk_2xMAC;

wire        clk_2xMACinta;
wire        clk_2xMAC;

IBUF   IBUF_MAC             (.I(clk_MAC_i),.O(clk_MACdlli));

CLKDLL DLL_MAC              (
                        .CLK0(clk_MACdllo),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(clk_2xMACdllo),
                        .CLKDV(),
                        .LOCKED(),
                        .CLKIN(clk_MACdlli),
                        .CLKFB(clk_MAC),
                        .RST(~sdmprj_reset_l)
                        );

BUFG  BUFG_MAC              (.I(clk_MACdllo),.O(clk_MAC));

BUFG  BUFG_2xMAC(.I(clk_2xMACdllo),.O(clk_2xMAC));


/*****************************************\
*               Chnnet 0 DLL              *
\*****************************************/
```

```
//--------------DLL CHNNET 0 ------------//

CLKDLL CHN0_DLL (
              .CLK0(sdmprj_chn0_clk_155),
              .CLK90(),
              .CLK180(sdmprj_chn0_clk_155_l),
              .CLK270(),
              .CLK2X(),
              .CLKDV(),
              .LOCKED(),
              .CLKIN(clk_2xMAC),
              .CLKFB(sdmprj_chn0_clk_155),
              .RST(~sdmprj_reset_l)
              );


//--------------DLL CHNNET 0 ------------//

CLKDLL CHN1_DLL (
              .CLK0(sdmprj_chn1_clk_155),
              .CLK90(),
              .CLK180(sdmprj_chn1_clk_155_l),
              .CLK270(),
              .CLK2X(),
              .CLKDV(),
              .LOCKED(),
              .CLKIN(clk_2xMAC),
              .CLKFB(sdmprj_chn1_clk_155),
              .RST(~sdmprj_reset_l)
                );


OBUF_LVPECL Psdmprj_chn0_clk_155        (.I(sdmprj_chn0_clk_155),    .O(sdmprj_chn0_clk_155_o));
OBUF_LVPECL Nsdmprj_chn0_clk_155        (.I(sdmprj_chn0_clk_155_l),  .O(sdmprj_chn0_clk_155_o));

OBUF_LVPECL Psdmprj_chn1_clk_155        (.I(sdmprj_chn1_clk_155),    .O(sdmprj_chn1_clk_155_o));
OBUF_LVPECL Nsdmprj_chn1_clk_155        (.I(sdmprj_chn1_clk_155_l),  .O(sdmprj_chn1_clk_155_o));



/*****************************************\
*            Chnnet 0 LVPECL             *
\*****************************************/

//CLK
IBUF_LVPECL Psdmprj_chn0_tclk_155        (.I(sdmprj_chn0_tclk_155_i),  .O(sdmprj_chn0_tclk_155));
IBUF_LVPECL Nsdmprj_chn0_tclk_155_l (.I(sdmprj_chn0_tclk_155_l_i), .O());

//DATA
IBUF_LVPECL P0sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[0]),  .O(sdmprj_chn0_tdat_p[0]));
IBUF_LVPECL P1sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[1]),  .O(sdmprj_chn0_tdat_p[1]));
IBUF_LVPECL P2sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[2]),  .O(sdmprj_chn0_tdat_p[2]));
IBUF_LVPECL P3sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[3]),  .O(sdmprj_chn0_tdat_p[3]));
IBUF_LVPECL P4sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[4]),  .O(sdmprj_chn0_tdat_p[4]));
IBUF_LVPECL P5sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[5]),  .O(sdmprj_chn0_tdat_p[5]));
IBUF_LVPECL P6sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[6]),  .O(sdmprj_chn0_tdat_p[6]));
IBUF_LVPECL P7sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[7]),  .O(sdmprj_chn0_tdat_p[7]));
IBUF_LVPECL P8sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[8]),  .O(sdmprj_chn0_tdat_p[8]));
IBUF_LVPECL P9sdmprj_chn0_tdatp  (.I(sdmprj_chn0_tdat_p_i[9]),  .O(sdmprj_chn0_tdat_p[9]));
IBUF_LVPECL P10sdmprj_chn0_tdatp         (.I(sdmprj_chn0_tdat_p_i[10]), .O(sdmprj_chn0_tdat_p[10]));
IBUF_LVPECL P11sdmprj_chn0_tdatp         (.I(sdmprj_chn0_tdat_p_i[11]), .O(sdmprj_chn0_tdat_p[11]));
IBUF_LVPECL P12sdmprj_chn0_tdatp            (.I(sdmprj_chn0_tdat_p_i[12]),  .O(sdmprj_chn0_tdat_p[12]));
IBUF_LVPECL P13sdmprj_chn0_tdatp            (.I(sdmprj_chn0_tdat_p_i[13]),  .O(sdmprj_chn0_tdat_p[13]));
IBUF_LVPECL P14sdmprj_chn0_tdatp            (.I(sdmprj_chn0_tdat_p_i[14]),  .O(sdmprj_chn0_tdat_p[14]));
IBUF_LVPECL P15sdmprj_chn0_tdatp            (.I(sdmprj_chn0_tdat_p_i[15]),  .O(sdmprj_chn0_tdat_p[15]));


/*****************************************\
*              Chnnet 1 LVPECL            *
\*****************************************/
//CLK
IBUF_LVPECL Psdmprj_chn1_tclk_155           (.I(sdmprj_chn1_tclk_155_i),  .O(sdmprj_chn1_tclk_155));
IBUF_LVPECL Nsdmprj_chn1_tclk_155_l (.I(sdmprj_chn1_tclk_155_l_i), .O());

//DATA
IBUF_LVPECL P0sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[0]),  .O(sdmprj_chn1_tdat_p[0]));
IBUF_LVPECL P1sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[1]),  .O(sdmprj_chn1_tdat_p[1]));
IBUF_LVPECL P2sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[2]),  .O(sdmprj_chn1_tdat_p[2]));
IBUF_LVPECL P3sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[3]),  .O(sdmprj_chn1_tdat_p[3]));
IBUF_LVPECL P4sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[4]),  .O(sdmprj_chn1_tdat_p[4]));
IBUF_LVPECL P5sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[5]),  .O(sdmprj_chn1_tdat_p[5]));
IBUF_LVPECL P6sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[6]),  .O(sdmprj_chn1_tdat_p[6]));
IBUF_LVPECL P7sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[7]),  .O(sdmprj_chn1_tdat_p[7]));
IBUF_LVPECL P8sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[8]),  .O(sdmprj_chn1_tdat_p[8]));
IBUF_LVPECL P9sdmprj_chn1_tdatp  (.I(sdmprj_chn1_tdat_p_i[9]),  .O(sdmprj_chn1_tdat_p[9]));
IBUF_LVPECL P10sdmprj_chn1_tdatp            (.I(sdmprj_chn1_tdat_p_i[10]),  .O(sdmprj_chn1_tdat_p[10]));
IBUF_LVPECL P11sdmprj_chn1_tdatp            (.I(sdmprj_chn1_tdat_p_i[11]),  .O(sdmprj_chn1_tdat_p[11]));
IBUF_LVPECL P12sdmprj_chn1_tdatp            (.I(sdmprj_chn1_tdat_p_i[12]),  .O(sdmprj_chn1_tdat_p[12]));
IBUF_LVPECL P13sdmprj_chn1_tdatp            (.I(sdmprj_chn1_tdat_p_i[13]),  .O(sdmprj_chn1_tdat_p[13]));
IBUF_LVPECL P14sdmprj_chn1_tdatp            (.I(sdmprj_chn1_tdat_p_i[14]),  .O(sdmprj_chn1_tdat_p[14]));
IBUF_LVPECL P15sdmprj_chn1_tdatp            (.I(sdmprj_chn1_tdat_p_i[15]),  .O(sdmprj_chn1_tdat_p[15]));


sdmproj_top usdmproj_top(
                              .clk_MAC(clk_MAC),
                              .clk_SYS(clk_SYS),
                              .sdmprj_reset_l(sdmprj_reset_l_i),
                              .sdmprj_mb_add_data(sdmprj_mb_add_data_io),
                              .sdmprj_mb_reset(sdmprj_mb_reset_i),
                              .sdmprj_mb_start(sdmprj_mb_start_i),
                              .sdmprj_mb_busy(sdmprj_mb_busy_o),
                              .sdmprj_mb_select(sdmprj_mb_select_i),
                              .sdmprj_mb_clk_en_l(sdmprj_mb_clk_en_l_i),
                              .sdmprj_mb_int_l(sdmprj_mb_int_l_o),
                              .sdmprj_txsel_l(sdmprj_txsel_o),
                              .sdmprj_fps_txf(sdmprj_fps_txf_o),
                              .sdmprj_sop_txf(sdmprj_sop_txf_o),
                              .sdmprj_eop_txf(sdmprj_eop_txf_o),
                              .sdmprj_vtg(sdmprj_vtg_o),
                              .sdmprj_txasis(sdmprj_txasis_o),
                              .sdmprj_flct(sdmprj_flct_o),
                              .sdmprj_flct_lat(sdmprj_flct_lat_o),
                              .sdmprj_txrdy_0(sdmprj_txrdy_0_i),
                              .sdmprj_txrdy_1(sdmprj_txrdy_1_i),
                              .sdmprj_chn0_tdat_p(sdmprj_chn0_tdat_p),
                              .sdmprj_chn0_tclk_155(sdmprj_chn0_tclk_155),
                              .sdmprj_chn0_tclk_155_l(),
                              .sdmprj_chn0_clk_155(),
                              .sdmprj_chn0_clk_155_l(),
                              .sdmprj_chn0_ext_rd,
                              .sdmprj_tx_si(sdmprj_tx_si_o),
                              .sdmprj_tx_ld(sdmprj_tx_ld_o),
                              .sdmprj_tx_sen_l(sdmprj_tx_sen_l_o),
.sdmprj_tx_rst_l(sdmprj_tx_rst_l_o),
.sdmprj_tx0_data(sdmprj_tx0_data_o),
.sdmprj_tx0_be(sdmprj_tx0_be_o),
.sdmprj_tx0_wclk(),
.sdmprj_tx0_ren(sdmprj_tx0_ren_o),
.sdmprj_tx0_oe_l(sdmprj_tx0_oe_l_o),
.sdmprj_tx0_wen(sdmprj_tx0_wen_o),
.sdmprj_tx0_ffl(sdmprj_tx0_ffl_i),
.sdmprj_tx0_efl(sdmprj_tx0_efl_i),
.sdmprj_tx0_pafl(sdmprj_tx0_pafl_i),
.sdmprj_chn1_tdat_p,
.sdmprj_chn1_tclk_155,
.sdmprj_chn1_tclk_155_l,
.sdmprj_chn1_clk_155,
.sdmprj_chn1_clk_155_l,
.sdmprj_chn1_ext_rd,
.sdmprj_tx1_data,
.sdmprj_tx1_be,
.sdmprj_tx1_wclk,
.sdmprj_tx1_ren,
.sdmprj_tx1_oe_l,
.sdmprj_tx1_wen,
.sdmprj_tx1_ffl,
.sdmprj_tx1_efl,
.sdmprj_tx1_pafl,
.sdmprj_vlanram_clk,
.sdmprj_vlanram_cs,
.sdmprj_vlanram_oe_l,
.sdmprj_vlanram_we_l,
.sdmprj_vlanram_adv,
.sdmprj_vlanram_addr,
.sdmprj_vlanram_data,
.sdmprj_ram_clk_mirror_in,
.sdmprj_statsram_clk,
.sdmprj_statsram_cs,
.sdmprj_statsram_oe_l,
.sdmprj_statsram_we_l,
.sdmprj_statsram_adv,
.sdmprj_statsram_addr,
.sdmprj_statsram_data,
.sdmprj_ram_clk_mirror_out
);
```

```
////////////////////////////////////////////////////////////
// File  : sdmproj_top.v
// Author: Roberto Acosta
//
// Description:
// This is the Top module for the FPGA
//
// All ports with suffix '_l' are low asserted.
////////////////////////////////////////////////////////////
// Revision History:
//
// Date                    Author   Revision
// Mar. 15th, 2009         R.A.     First Pass
//
//
////////////////////////////////////////////////////////////

`timescale 1ns/10 ps

// synopsys translate_off

//`include "XilinxChaneLib/async_fifo_v1_0.v"

// synopsys translate_on


module sdmproj_top          (
                            clk_MACi,
                            clk_SYSi,
                            sdmproj_reset_l,
                            sdmproj_mb_add_data,
                            sdmproj_mb_reset,
                            sdmproj_mb_start,
                            sdmproj_mb_busy,
                            sdmproj_mb_select,
                            sdmproj_mb_clk_en_l,
                            sdmproj_mb_int_l,
                            sdmproj_txsel_l,
                            sdmproj_fps_txf,
                            sdmproj_sop_txf,
                            sdmproj_eop_txf,
                            sdmproj_vtg,
                            sdmproj_txasis,
                            sdmproj_flct,
                            sdmproj_flct_lat,
                            sdmproj_txrdy_0,
                            sdmproj_txrdy_1,
                            sdmproj_chan0_tdat_p,
                            sdmproj_chan0_tclk_155,
                            sdmproj_chan0_tclk_155_l,
                            sdmproj_chan0_clk_155,
                            sdmproj_chan0_clk_155_l,
                            sdmproj_chan0_ext_rd,
                            sdmproj_tx_si,
                                sdmproj_tx_ld,
                                sdmproj_tx_sen_l,
                                sdmproj_tx_rst_l,
                                sdmproj_tx0_data,
                                sdmproj_tx0_be,
                                sdmproj_tx0_wclk,
                                sdmproj_tx0_ren,
                                sdmproj_tx0_oe_l,
                                sdmproj_tx0_wen,
                                sdmproj_tx0_ffl,
                                sdmproj_tx0_efl,
                                sdmproj_tx0_pafl,
                                sdmproj_chan1_tdat_p,
                                sdmproj_chan1_tclk_155,
                                sdmproj_chan1_tclk_155_l,
                                sdmproj_chan1_clk_155,
                                sdmproj_chan1_clk_155_l,
                                sdmproj_chan1_ext_rd,
                                sdmproj_tx1_data,
                                sdmproj_tx1_be,
                                sdmproj_tx1_wclk,
                                sdmproj_tx1_ren,
                                sdmproj_tx1_oe_l,
                                sdmproj_tx1_wen,
                                sdmproj_tx1_ffl,
                                sdmproj_tx1_efl,
                                sdmproj_tx1_pafl,
                                sdmproj_vlanram_clk,
                                sdmproj_vlanram_cs,
                                sdmproj_vlanram_oe_l,
                                sdmproj_vlanram_we_l,
                                sdmproj_vlanram_adv,
                                sdmproj_vlanram_addr,
                                sdmproj_vlanram_data,
                                sdmproj_ram_clk_mirror_in,
                                sdmproj_statsram_clk,
                                sdmproj_statsram_cs,
                                sdmproj_statsram_oe_l,
                                sdmproj_statsram_we_l,
                                sdmproj_statsram_adv,
                                sdmproj_statsram_addr,
                                sdmproj_statsram_data,
                                sdmproj_ram_clk_mirror_out,
                                sdmproj_spare1,
                                sdmproj_spare2,
                                sdmproj_tx0_wclk_mirror,
                                sdmproj_tx1_wclk_mirror,
                                sdmproj_port0_ext_fifo_rst,
                                sdmproj_port1_ext_fifo_rst,
                                sdmproj_flow_ctl_p0,
                                sdmproj_flow_ctl_p1
//                              sdmproj_debug_lockout
                                );
parameter TPD = 1;

//////////////////////////////////////////
```

```
// SYSTEM
///////////////////////////////////////////

input              clk_MACi;
input              clk_SYSi;
input              sdmproj_reset_l;

///////////////////////////////////////////
// MAINTENANCE BUS
///////////////////////////////////////////

inout [17:0]       sdmproj_mb_add_data;
input              sdmproj_mb_reset;
input              sdmproj_mb_start;
input              sdmproj_mb_select;
input              sdmproj_mb_clk_en_l;
output             sdmproj_mb_busy;
output[1:0]        sdmproj_mb_int_l;

///////////////////////////////////////////
// SPARE
///////////////////////////////////////////

output   [16:0] sdmproj_spare1;
output   [16:0] sdmproj_spare2;
//output  sdmproj_debug_lockout;

///////////////////////////////////////////
// Common Port Fifo Interface
///////////////////////////////////////////

output             sdmproj_tx_si;
output             sdmproj_tx_ld;
output             sdmproj_tx_sen_l;
output             sdmproj_tx_rst_l;

///////////////////////////////////////////
// Port 1 Interfaces
///////////////////////////////////////////
// Channet

input [15:0]       sdmproj_chan0_tdat_p;
input              sdmproj_chan0_tclk_155;
input              sdmproj_chan0_tclk_155_l;
output             sdmproj_chan0_clk_155;
output             sdmproj_chan0_clk_155_l;
output             sdmproj_chan0_ext_rd;

//FIFO

output[31:0]       sdmproj_tx0_data;
output[3:0]        sdmproj_tx0_be;
output             sdmproj_tx0_wclk;
output             sdmproj_tx0_ren;
output             sdmproj_tx0_oe_l;
output             sdmproj_tx0_wen;
```

```
input           sdmproj_tx0_ffl;
input           sdmproj_tx0_efl;
input           sdmproj_tx0_pafl;

input           sdmproj_tx0_wclk_mirror; //RA
output          sdmproj_port0_ext_fifo_rst;
///////////////////////////////////////////
// Port 2 Interfaces
///////////////////////////////////////////
// Channel 1

input [15:0]    sdmproj_chan1_tdat_p;
input           sdmproj_chan1_tclk_155;
input           sdmproj_chan1_tclk_155_l;
output          sdmproj_chan1_clk_155;
output          sdmproj_chan1_clk_155_l;
output          sdmproj_chan1_ext_rd;

//FIFO

output[31:0]    sdmproj_tx1_data;
output[3:0]     sdmproj_tx1_be;
output          sdmproj_tx1_wclk;
output          sdmproj_tx1_ren;
output          sdmproj_tx1_oe_l;
output          sdmproj_tx1_wen;
input           sdmproj_tx1_ffl;
input           sdmproj_tx1_efl;
input           sdmproj_tx1_pafl;

input           sdmproj_tx1_wclk_mirror; //RA
output          sdmproj_port1_ext_fifo_rst;
///////////////////////////////////////////
// Memory Interface (L2 Table)
///////////////////////////////////////////

output          sdmproj_vlanram_clk;
output          sdmproj_vlanram_cs;
output          sdmproj_vlanram_oe_l;
output          sdmproj_vlanram_we_l;
output          sdmproj_vlanram_adv;
output[18:0]    sdmproj_vlanram_addr;
inout [35:0]    sdmproj_vlanram_data;
input           sdmproj_ram_clk_mirror_in;

///////////////////////////////////////////
// Memory Interface (Counters)
///////////////////////////////////////////

output          sdmproj_statsram_clk;
output          sdmproj_statsram_cs;
output          sdmproj_statsram_oe_l;
output          sdmproj_statsram_we_l;
output          sdmproj_statsram_adv;
output[18:0]    sdmproj_statsram_addr;
inout [35:0]    sdmproj_statsram_data;
```

```
input           sdmproj_ram_clk_mirror_out;

////////////////////////////////////////////
// MAC Interface
////////////////////////////////////////////
input           sdmproj_flow_ctl_p0;
input           sdmproj_flow_ctl_p1;

output          sdmproj_txsel_l;
output          sdmproj_fps_txf /* synthesis syn_useioff = 1 */;
output          sdmproj_sop_txf;
output          sdmproj_eop_txf;
output          sdmproj_vtg;
output          sdmproj_txasis;
output[1:0]     sdmproj_flct;
output          sdmproj_flct_lat;
input           sdmproj_txrdy_0;
input           sdmproj_txrdy_1;



////////////////////////////////////////////
// Inter-Module Wires
////////////////////////////////////////////

/**************Clocks MAIN*****************/
wire    clk_SYSi;
wire    clk_SYSint;
wire    clk_SYSinta;
wire    clk_SYS /* synthesis syn_keep=1 */;

wire    clk_2xSYSa;
wire    clk_2xSYS  /* synthesis syn_keep=1 */;
wire    locked_2xSYS;
wire    SRL_output;
wire    SRL_output_not;

// Debug signals for dll locks jw
//wire  sdmproj_debug_lockout /* synthesis syn_keep=1 syn_preserve=1 */;
//wire  clk0_oe /* synthesis syn_keep=1 syn_preserve=1 */;
//wire  clk1_oe /* synthesis syn_keep=1 syn_preserve=1 */;

`ifdef VERA_SIM_ENV
assign  SRL_output_not   = ~sdmproj_reset_l;
`else
assign  SRL_output_not = ~SRL_output;
`endif

//assign  SRL_output_not = ~sdmproj_reset_l;

wire     logic1;
assign  logic1 = 1'b1;


`ifdef VERA_SIM_ENV
`else
```

```
IBUFG  I_ibufg_clk_SYS(
                        .I(clk_SYSi),
                        .O(clk_SYSint)
                        );
`endif

`ifdef VERA_SIM_ENV
CLKDLL usdmproj_clkdllSYS(
                        .CLK0(clk_SYSinta),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(clk_2xSYSa),
                        .CLKDV(),
                        .LOCKED(locked_2xSYS),
                        .CLKIN(clk_SYSi),
                        .CLKFB(clk_SYS),
                        .RST()
                        );
`else
CLKDLL usdmproj_clkdllSYS(
                        .CLK0(clk_SYSinta),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(clk_2xSYSa),
                        .CLKDV(),
                        .LOCKED(locked_2xSYS),
                        .CLKIN(clk_SYSint),
                        .CLKFB(clk_2xSYS),
                        .RST()
                        );
`endif

SRL16 usdmproj_srl16SYS (
                        .Q(SRL_output),
                        .A0(logic1),
                        .A1(logic1),
                        .A2(logic1),
                        .A3(logic1),
                        .D(locked_2xSYS),
                        .CLK(clk_2xSYS)
                        );

BUFG  usdmproj_bufgSYS          (
                        .I(clk_SYSinta),
                        .O(clk_SYS)
                        );

BUFG  usdmproj_bufg2xSYS(
                        .I(clk_2xSYSa),
                        .O(clk_2xSYS)
                        );


wire            sdmproj_tx0_wclkinta;
wire            sdmproj_tx1_wclkinta;
wire            sdmproj_tx0_wclk;
wire            sdmproj_tx1_wclk;
wire            sdmproj_tx0_wclk_mirror;
wire            sdmproj_tx1_wclk_mirror;

wire    sdmproj_tx0_locked;
wire    sdmproj_tx1_locked;

//assign sdmproj_tx1_wclk = sdmproj_tx0_wclk;

`ifdef VERA_SIM_ENV
CLKDLL usdmproj_clkdlltx0(
                        .CLK0(sdmproj_tx0_wclkinta),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(),
                        .CLKDV(),
                        .LOCKED(sdmproj_tx0_locked),
                        .CLKIN(clk_SYSi),
                        .CLKFB(clk_SYS),
                        .RST()
                        );
`else
CLKDLL usdmproj_clkdlltx0(
                        .CLK0(sdmproj_tx0_wclkinta),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(),
                        .CLKDV(),
                        .LOCKED(sdmproj_tx0_locked),
                        .CLKIN(clk_SYSint),
                        .CLKFB(sdmproj_tx0_wclk_mirror),
                        .RST()
                        );
`endif
OBUF_F_12 I_obuf_tx0_wclk (
                         .I(sdmproj_tx0_wclkinta),
                         .O(sdmproj_tx0_wclk)
                         );
`ifdef VERA_SIM_ENV
CLKDLL usdmproj_clkdlltx1(
                        .CLK0(sdmproj_tx1_wclkinta),
                        .CLK90(),
                        .CLK180(),
                        .CLK270(),
                        .CLK2X(),
                        .CLKDV(),
                        .LOCKED(sdmproj_tx1_locked),
                        .CLKIN(clk_SYSi),
                        .CLKFB(clk_SYS),
                        .RST()
                        );
`else
```

```
CLKDLL usdmproj_clkdlltx1(
                              .CLK0(sdmproj_tx1_wclkinta),
                              .CLK90(),
                              .CLK180(),
                              .CLK270(),
                              .CLK2X(),
                              .CLKDV(),
                              .LOCKED(sdmproj_tx1_locked),
                              .CLKIN(clk_SYSint),
                              .CLKFB(sdmproj_tx1_wclk_mirror),
                              .RST()
                              );
`endif

OBUF_F_12 I_obuf_tx1_wclk (
                               .I(sdmproj_tx1_wclkinta),
                               .O(sdmproj_tx1_wclk)
                               );

wire   clk_2xSYSo;          //Wire between DLL and OBUFG (int-ext)
wire   clk_2xSYSb;          //Wire between DLL and OBUFG (int-ext)
wire     sdmproj_vlanram_clk;
wire     sdmproj_statsram_clk;
wire   sdmproj_vlan_locked;

wire   sdmproj_ram_clk_mirror_in_int;

IBUFG ram_mirror (
                       .I(sdmproj_ram_clk_mirror_in),
                       .O(sdmproj_ram_clk_mirror_in_int )
                      );


`ifdef VERA_SIM_ENV
CLKDLL usdmproj_clkdll_vlan(
                               .CLK0(),
                               .CLK90(),
                               .CLK180(),
                               .CLK270(),
                               .CLK2X(clk_2xSYSo),
                               .CLKDV(),
                               .LOCKED(sdmproj_vlan_locked),
                               .CLKIN(clk_SYSi),
                               .CLKFB(clk_SYS),
                               .RST()
                               );

`else
CLKDLL usdmproj_clkdll_vlan(
                               .CLK0(),
                               .CLK90(),
                               .CLK180(),
                               .CLK270(),
                               .CLK2X(clk_2xSYSo),
                               .CLKDV(),
                               .LOCKED(sdmproj_vlan_locked),
                    .CLKIN(clk_SYSint),
                    .CLKFB(sdmproj_ram_clk_mirror_in_int),
                    .RST()
                    );

`endif


OBUF_F_12 I_obuf_clk2x_vlan (
                    .I(clk_2xSYSo),
                    .O(sdmproj_vlanram_clk)
                     );

OBUF_F_12 I_obuf_clk2x_stats (
                    .I(clk_2xSYSo),
                    .O(sdmproj_statsram_clk)
                     );


/**************Clocks 77*****************/
wire    clk_MACi;
wire    clk_MACint;
wire    clk_MACinta;
wire    clk_MAC /* synthesis syn_keep=1 */;


wire    clk_2xMACinta;
wire    clk_2xMAC  /* synthesis syn_keep=1 */;
wire    locked_2xMAC;
wire    SRLMAC_output;
wire    SRLMAC_output_not;

`ifdef VERA_SIM_ENV
`else
IBUFG  I_ibufg_clk_MAC          (
                    .I(clk_MACi),
                    .O(clk_MACint)
                    );
`endif

`ifdef VERA_SIM_ENV
CLKDLL usdmproj_clkdllMAC(
                    .CLK0(clk_MACinta),
                    .CLK90(),
                    .CLK180(),
                    .CLK270(),
                    .CLK2X(clk_2xMACinta),
                    .CLKDV(),
                    .LOCKED(locked_2xMAC),
                    .CLKIN(clk_MACi),
                    .CLKFB(clk_MACinta),
                    .RST()
                    );


BUFG  usdmproj_bufgMAC          (
                               .I(clk_MACi),
                               .O(clk_MAC)
                               );
`else
CLKDLL usdmproj_clkdllMAC(
                               .CLK0(clk_MACinta),
                               .CLK90(),
                               .CLK180(),
                               .CLK270(),
                               .CLK2X(clk_2xMACinta),
                               .CLKDV(),
                               .LOCKED(locked_2xMAC),
                               .CLKIN(clk_MACint),
                               .CLKFB(clk_MACinta),
                               .RST()
                               );


BUFG  usdmproj_bufgMAC         (
                               .I(clk_MACinta),
                               .O(clk_MAC)
                               );
`endif

/**************Clocks 155*****************/
/*
wire sdmproj_chan0_clk_155inta;
wire sdmproj_chan0_clk_155_linta;
wire sdmproj_chan0_clk_155;
assign    sdmproj_chan0_clk_155 = clk_2xSYS;
wire sdmproj_chan0_clk_155_l;
assign sdmproj_chan0_clk_155_l = ~clk_2xSYS;

wire sdmproj_chan1_clk_155inta;
wire sdmproj_chan1_clk_155_linta;
wire sdmproj_chan1_clk_155;
assign    sdmproj_chan1_clk_155 = clk_2xSYS;
wire sdmproj_chan1_clk_155_l;
assign sdmproj_chan1_clk_155_l = ~clk_2xSYS;
*/
/***********Channet Clocks*******************/
IBUF_LVPECL data0_p (.I(sdmproj_chan0_tclk_155), .O(sdmproj_chan0_tclk_155_int));
IBUF_LVPECL data0_n (.I(sdmproj_chan0_tclk_155_l), .O());

IBUF_LVPECL data1_p (.I(sdmproj_chan1_tclk_155), .O(sdmproj_chan1_tclk_155_int));
IBUF_LVPECL data1_n (.I(sdmproj_chan1_tclk_155_l), .O());


/***********Local Reg Module************/
//SAXE
wire [17:0] sdmproj_mb_add_data;
wire        sdmproj_mb_reset;
wire        sdmproj_mb_start;
wire        sdmproj_mb_busy;
wire        sdmproj_mb_select;
wire        sdmproj_mb_clk_en_l;
wire [1:0]  sdmproj_mb_int_l;


//ZBT MODULE (LOOKUP)
wire [20:0]        sdmproj_mb_zbt_add_out;
wire [35:0]        sdmproj_mb_zbt_data_out;
wire               sdmproj_mb_zbt_we;
wire               sdmproj_mb_zbt_valid;
wire [31:0]        sdmproj_mb_zbt_data_in;
wire               sdmproj_mb_zbt_rd;
wire               mb_clk_en;

//ZBT MODULE (STATS)
wire [20:0]        sdmproj_mb_zbt_add_out;

wire               sdmproj_mb_stats_we;
wire               sdmproj_mb_stats_valid;
wire [31:0]        sdmproj_mb_stats_data_in;
wire               sdmproj_mb_stats_rd;


//PIPE 0
wire         sdmproj_port0_special;
wire   [7:0]  sdmproj_port0_delay;
wire   [3:0]  sdmproj_port0_mode;
wire         sdmproj_port0_link_state;
             assign sdmproj_port0_link_state = sdmproj_port0_mode[3];
wire   [1:0]  sdmproj_port0_int_en;
wire  [47:0]  sdmproj_port0_sa;
wire  [31:0]  sdmproj_port0_mpls_tag;
wire [127:0]  sdmproj_port0_l2_table;
wire  [31:0]  sdmproj_port0_pkts_count;
wire  [31:0]  sdmproj_port0_pkts_count_sop;
wire  [31:0]  sdmproj_port0_oct_count;
wire         sdmproj_port0_pkts_clear;
wire         sdmproj_port0_pkts_clear_sop;
wire  [31:0]  sdmproj_mfifo0_pkts_count_eop;
wire  [31:0]  sdmproj_mfifo0_pkts_count_sop;
wire         sdmproj_mfifo0_pkts_count_eop_clear;
wire         sdmproj_mfifo0_pkts_count_sop_clear;
wire  [31:0]  sdmproj_ipipe0_pkts_count_eop;
wire  [31:0]  sdmproj_ipipe0_pkts_count_sop;
wire         sdmproj_ipipe0_pkts_count_eop_clear;
wire         sdmproj_ipipe0_pkts_count_sop_clear;
wire  [31:0]  sdmproj_discard0_sop_count;
wire  [31:0]  sdmproj_discard0_eop_count;
wire         sdmproj_discard0_sop_clear;
wire         sdmproj_discard0_eop_clear;

wire  [15:0]  sdmproj_port0_link_upcount;
wire        sdmproj_port0_link_upclear;
wire  [15:0]  sdmproj_port0_link_dncount;
wire         sdmproj_port0_link_dnclear;

wire         sdmproj_port0_l2_error;
wire         sdmproj_port0_overflow;
wire          sdmproj_port0_zbt_error;
wire          sdmproj_port0_l2_error_clear;
wire          sdmproj_port0_overflow_clear;
wire          sdmproj_port0_zbt_error_clear;

wire          sdmproj_port0_vlan_rd;
wire  [31:0]  sdmproj_port0_vlan_dout;
wire          sdmproj_port0_vlan_valid;

wire  [31:0]  debug_bits;

//PIPE 1
wire          sdmproj_port1_special;
wire   [7:0]  sdmproj_port1_delay;
wire   [3:0]  sdmproj_port1_mode;
wire          sdmproj_port1_link_state;
              assign sdmproj_port1_link_state = sdmproj_port1_mode[3];
wire   [1:0]  sdmproj_port1_int_en;
wire  [47:0]  sdmproj_port1_sa;
wire  [31:0]  sdmproj_port1_mpls_tag;
wire [127:0]  sdmproj_port1_l2_table;
wire  [31:0]  sdmproj_port1_pkts_count;
wire  [31:0]  sdmproj_port1_pkts_count_sop;
wire  [31:0]  sdmproj_port1_oct_count;
wire          sdmproj_port1_pkts_clear;
wire          sdmproj_port1_pkts_clear_sop;
wire  [31:0]  sdmproj_mfifo1_pkts_count_eop;
wire  [31:0]  sdmproj_mfifo1_pkts_count_sop;
wire          sdmproj_mfifo1_pkts_count_eop_clear;
wire          sdmproj_mfifo1_pkts_count_sop_clear;
wire  [31:0]  sdmproj_ipipe1_pkts_count_eop;
wire  [31:0]  sdmproj_ipipe1_pkts_count_sop;
wire          sdmproj_ipipe1_pkts_count_eop_clear;
wire          sdmproj_ipipe1_pkts_count_sop_clear;
wire  [31:0]  sdmproj_discard1_sop_count;
wire  [31:0]  sdmproj_discard1_eop_count;
wire          sdmproj_discard1_sop_clear;
wire          sdmproj_discard1_eop_clear;

wire  [15:0]  sdmproj_port1_link_upcount;
wire        sdmproj_port1_link_upclear;
wire  [15:0]  sdmproj_port1_link_dncount;
wire        sdmproj_port1_link_dnclear;

wire          sdmproj_port1_l2_error;
wire          sdmproj_port1_overflow;
wire          sdmproj_port1_zbt_error;
wire          sdmproj_port1_l2_error_clear;
wire          sdmproj_port1_overflow_clear;
wire          sdmproj_port1_zbt_error_clear;

wire          sdmproj_port1_vlan_rd;
wire  [31:0]  sdmproj_port1_vlan_dout;
wire          sdmproj_port1_vlan_valid;

/***********PIPE 0 Module************/
```

```
//IN
wire [15:0]         sdmproj_chan0_tdat_p;
wire                sdmproj_chan0_ext_rd;
wire                sdmproj_chan0_tclk_155;
wire                sdmproj_chan0_tclk_155_l;

//OUT
wire [31:0]         sdmproj_tx0_data;
wire [3:0]          sdmproj_tx0_be;
wire                sdmproj_tx0_wen;
wire                sdmproj_tx0_wen_i;
wire                sdmproj_tx0_pafl;

//With Internal Storage fifos
wire                sdmproj_tx0_sop;
wire                sdmproj_tx0_eop;
wire                sdmproj_tx0_abr;
wire                sdmproj_tx0_pafl_int;

//With External Storage fifos
wire                sdmproj_tx_ld;
                assign sdmproj_tx_ld = 1'b1;
wire                sdmproj_tx_rst_l;

//With ZBT RAM (LOOKUP)
wire [15:0]   sdmproj_p0_zbt_add_out;
wire          sdmproj_p0_zbt_we;
wire [31:0]   sdmproj_p0_zbt_data_out;
wire        sdmproj_p0_zbt_rd;
wire          sdmproj_p0_zbt_valid;
wire          sdmproj_p0_zbt_afull;

//With Mictor
wire [16:0]   sdmproj_port0_spare1;
wire [16:0]   sdmproj_port0_spare2;

/***********PIPE 1 Module************/
//IN
wire [15:0]         sdmproj_chan1_tdat_p;
wire                sdmproj_chan1_ext_rd;
wire                sdmproj_chan1_tclk_155;
wire                sdmproj_chan1_tclk_155_l;

//OUT
wire [31:0]         sdmproj_tx1_data;
wire  [3:0]         sdmproj_tx1_be;
wire                sdmproj_tx1_wen;
wire                sdmproj_tx1_wen_i;
wire                sdmproj_tx1_pafl;

//With Internal Storage fifos
wire                sdmproj_tx1_sop;
wire                sdmproj_tx1_eop;
wire                sdmproj_tx1_abr;
wire                sdmproj_tx1_pafl_int;
```

```
//With External Storage fifos
wire                sdmproj_tx_si;
                    assign sdmproj_tx_si = 1'b0;
wire                sdmproj_tx_sen_l;
                    assign sdmproj_tx_sen_l = 1'b1;

//With ZBT RAM (LOOKUP)
wire [15:0]  sdmproj_p1_zbt_add_out;
wire         sdmproj_p1_zbt_we;
wire [31:0]  sdmproj_p1_zbt_data_out;
wire       sdmproj_p1_zbt_rd;
wire         sdmproj_p1_zbt_valid;
wire         sdmproj_p1_zbt_afull;

//With Mictor
wire [16:0]  sdmproj_port1_spare1;
wire [16:0]  sdmproj_port1_spare2;

/***********MAC Module ************/
//SYS

wire     sdmproj_mac_p0_en;
         assign sdmproj_mac_p0_en = sdmproj_port0_int_en[1];

wire     sdmproj_mac_p1_en;
         assign sdmproj_mac_p1_en = sdmproj_port1_int_en[1];

//With Internal Storage fifos 0
wire     sdmproj_extfifo0_sop;
wire     sdmproj_extfifo0_eop;
wire     sdmproj_extfifo0_abr;

//With External Storage fifos 0
wire     sdmproj_tx0_ren;
wire     sdmproj_tx0_oe_l;
wire     sdmproj_tx0_sop_efl;
wire     sdmproj_port0_ext_fifo_rst;
assign  sdmproj_port0_ext_fifo_rst = sdmproj_port0_link_state;

//With Internal Storage fifos 1
wire     sdmproj_extfifo1_sop;
wire     sdmproj_extfifo1_eop;
wire     sdmproj_extfifo1_abr;

//With External Storage fifos 1
wire     sdmproj_tx1_ren;
wire     sdmproj_tx1_oe_l;
wire     sdmproj_tx1_sop_efl;
wire     sdmproj_port1_ext_fifo_rst;
assign  sdmproj_port1_ext_fifo_rst = sdmproj_port1_link_state;

//With MAC
wire     sdmproj_txsel_l;
wire     sdmproj_fps_txf;
wire     sdmproj_sop_txf;
wire     sdmproj_eop_txf;
wire     sdmproj_vtg;
wire     sdmproj_txasis;
wire[1:0]          sdmproj_flct;
wire     sdmproj_flct_lat;
wire     sdmproj_txrdy_0;
wire     sdmproj_txrdy_1;
wire     sdmproj_flow_ctl_p0;
wire     sdmproj_flow_ctl_p1;

/***********ZBT_RAM ************/

wire     sdmproj_vlanram_cs;
wire     sdmproj_vlanram_oe_l;
wire     sdmproj_vlanram_we_l;
wire     sdmproj_vlanram_adv;
wire [18:0]        sdmproj_vlanram_addr;
wire [35:0]        sdmproj_vlanram_data;
wire     sdmproj_ram_clk_mirror_in;



////////////////////////////////////////////
// Data Path for Ports
////////////////////////////////////////////


//wire    sdmproj_ipipe0_ctl_en;
//  assign sdmproj_ipipe0_ctl_en = sdmproj_port0_int_en[0];

//wire    sdmproj_ipipe1_ctl_en;
//  assign sdmproj_ipipe1_ctl_en = sdmproj_port1_int_en[0];

wire sdmproj_mac_ctl_en;
assign sdmproj_mac_ctl_en  = 1'b1;



/***************************************************************************\
* Sdmproj spare register connections                                        *
\***************************************************************************/
reg [15:0]         sdmproj_reg_spare1;
reg [15:0]         sdmproj_reg_spare2;
wire [16:0]        sdmproj_spare1;
wire [16:0]        sdmproj_spare2;

always @ (posedge clk_SYS or negedge sdmproj_reset_l)
begin
        if(~sdmproj_reset_l)
         begin
                sdmproj_reg_spare1 <= 16'h0000;
                sdmproj_reg_spare2 <= 16'h0000;
         end
        else
         begin
                sdmproj_reg_spare1 <= sdmproj_port0_mode[2] ? sdmproj_port0_spare1[15:0] :
sdmproj_port1_spare1[15:0];
```

```
                    sdmproj_reg_spare2 <= sdmproj_port0_mode[2] ? sdmproj_port0_spare2[15:0] :
sdmproj_port1_spare2[15:0];
          end
end

assign   sdmproj_spare1[16:0] = {clk_SYS,sdmproj_reg_spare1[15:0]};
assign  sdmproj_spare2[16:0] = {clk_SYS,sdmproj_reg_spare2[15:0]};

/***************************************************************************\
* Sdmproj fifo Partial Reset                                              *
\***************************************************************************/

/***************************************************************************\
* Wait for all DLLs to lock before outputing clocks                          *
\***************************************************************************/

//assign sdmproj_debug_lockout =  locked_2xSYS;

//assign clk0_oe = locked_2xMAC & locked_2xSYS;
//assign clk1_oe = locked_2xMAC & locked_2xSYS;


/**************Clocks 155*****************/

wire sdmproj_chan0_clk_155inta;
wire sdmproj_chan0_clk_155_linta;
wire sdmproj_chan0_clk_155;
wire sdmproj_chan0_clk_155_internal;
wire sdmproj_chan0_clk_155_n_internal;

//assign  sdmproj_chan0_clk_155_internal = sdmproj_port0_mode[1] ? clk_2xMACinta : clk_2xSYS;

assign    sdmproj_chan0_clk_155_internal = clk_2xSYS;


//OBUFT_LVPECL clk0_p (.I(sdmproj_chan0_clk_155_internal),   .T(clk0_oe), .O(sdmproj_chan0_clk_155));
//INV clk0_inv       (.I(sdmproj_chan0_clk_155_internal),   .O(sdmproj_chan0_clk_155_n_internal));
//OBUFT_LVPECL clk0_n (.I(sdmproj_chan0_clk_155_n_internal), .T(clk0_oe), .O(sdmproj_chan0_clk_155_l));

OBUF_LVPECL clk0_p (.I(sdmproj_chan0_clk_155_internal),   .O(sdmproj_chan0_clk_155));
INV clk0_inv      (.I(sdmproj_chan0_clk_155_internal),   .O(sdmproj_chan0_clk_155_n_internal));
OBUF_LVPECL clk0_n (.I(sdmproj_chan0_clk_155_n_internal), .O(sdmproj_chan0_clk_155_l));


wire sdmproj_chan1_clk_155inta;
wire sdmproj_chan1_clk_155_linta;
wire sdmproj_chan1_clk_155;
wire sdmproj_chan1_clk_155_internal;
wire sdmproj_chan1_clk_155_n_internal;


//assign  sdmproj_chan1_clk_155_internal  = sdmproj_port1_mode[1] ? clk_2xMACinta : clk_2xSYS;

assign    sdmproj_chan1_clk_155_internal = clk_2xSYS;

//OBUFT_LVPECL clk1_p (.I(sdmproj_chan1_clk_155_internal),   .T(clk1_oe), .O(sdmproj_chan1_clk_155));
```

```
//INV clk1_inv      (.I(sdmproj_chan1_clk_155_internal),  .O(sdmproj_chan1_clk_155_n_internal));
//OBUFT_LVPECL clk1_n (.I(sdmproj_chan1_clk_155_n_internal), .T(clk1_oe), .O(sdmproj_chan1_clk_155_l));

OBUF_LVPECL clk1_p (.I(sdmproj_chan1_clk_155_internal),  .O(sdmproj_chan1_clk_155));
INV clk1_inv      (.I(sdmproj_chan1_clk_155_internal), .O(sdmproj_chan1_clk_155_n_internal));
OBUF_LVPECL clk1_n (.I(sdmproj_chan1_clk_155_n_internal), .O(sdmproj_chan1_clk_155_l));


/*************************************************************************\
*       This is the Sdmproj Local Register module it also interfaces with *
* the NT SRAM and the VLAN NT SRAM                                        *
\*************************************************************************/
wire    reset_port1;
wire    reset_port0;
wire    reset_mac;

sdmproj_localreg   usdmproj_localreg   (
                                        .clk_SYS(clk_SYS),
                                    .sdmproj_reset_l(sdmproj_reset_l),
                                        .mb_clk_en_l(sdmproj_mb_clk_en_l),
                                        .mb_clk_en(mb_clk_en),
                                        .mb_reset(sdmproj_mb_reset),
                                        .mb_start(sdmproj_mb_start),
                                        .mb_select(sdmproj_mb_select),
                                        .mb_busy(sdmproj_mb_busy),
                                        .mb_add_data(sdmproj_mb_add_data),
                                        .sdmproj_mb_int_l(sdmproj_mb_int_l),
                                        .sdmproj_debug_bits(debug_bits),
                                        .sdmproj_tx0_pafl_int(sdmproj_tx0_pafl_int),
                                        .sdmproj_port0_special(sdmproj_port0_special),
                                        .sdmproj_port0_delay(sdmproj_port0_delay),
                                .sdmproj_port0_mode(sdmproj_port0_mode),
                                        .sdmproj_port0_int_en(sdmproj_port0_int_en),
                                        .sdmproj_port0_sa(sdmproj_port0_sa),
                                        .sdmproj_port0_mpls_tag(sdmproj_port0_mpls_tag),
                                        .sdmproj_port0_l2_table(sdmproj_port0_l2_table),
                                        .sdmproj_port0_pkts_count(sdmproj_port0_pkts_count),
                                        .sdmproj_port0_pkts_count_sop(sdmproj_port0_pkts_count_sop),
                                        .sdmproj_port0_oct_count(sdmproj_port0_oct_count),
                                        .sdmproj_port0_pkts_clear(sdmproj_port0_pkts_clear),
                                        .sdmproj_port0_pkts_clear_sop(sdmproj_port0_pkts_clear_sop),
                                        .sdmproj_port0_oct_clear(sdmproj_port0_oct_clear),
                                    .sdmproj_mfifo0_pkts_count_eop(sdmproj_mfifo0_pkts_count_eop),
                                    .sdmproj_mfifo0_pkts_count_sop(sdmproj_mfifo0_pkts_count_sop),

            .sdmproj_mfifo0_pkts_count_eop_clear(sdmproj_mfifo0_pkts_count_eop_clear),

            .sdmproj_mfifo0_pkts_count_sop_clear(sdmproj_mfifo0_pkts_count_sop_clear),
                                    .sdmproj_ipipe0_pkts_count_eop(sdmproj_ipipe0_pkts_count_eop),
                                    .sdmproj_ipipe0_pkts_count_sop(sdmproj_ipipe0_pkts_count_sop),

            .sdmproj_ipipe0_pkts_count_eop_clear(sdmproj_ipipe0_pkts_count_eop_clear),

            .sdmproj_ipipe0_pkts_count_sop_clear(sdmproj_ipipe0_pkts_count_sop_clear),
                                        .sdmproj_discard0_sop_count(sdmproj_discard0_sop_count),
                                        .sdmproj_discard0_eop_count(sdmproj_discard0_eop_count),
                                        .sdmproj_discard0_sop_clear(sdmproj_discard0_sop_clear),
                                        .sdmproj_discard0_eop_clear(sdmproj_discard0_eop_clear),
                                        .sdmproj_port0_link_upcount(sdmproj_port0_link_upcount),
                                        .sdmproj_port0_link_upclear(sdmproj_port0_link_upclear),
                                        .sdmproj_port0_link_dncount(sdmproj_port0_link_dncount),
                                        .sdmproj_port0_link_dnclear(sdmproj_port0_link_dnclear),
                                        .sdmproj_tx1_pafl_int(sdmproj_tx1_pafl_int),
                                        .sdmproj_port1_special(sdmproj_port1_special),
                                        .sdmproj_port1_delay(sdmproj_port1_delay),
                                        .sdmproj_port1_mode(sdmproj_port1_mode),
                                        .sdmproj_port1_int_en(sdmproj_port1_int_en),
                                        .sdmproj_port1_sa(sdmproj_port1_sa),
                                        .sdmproj_port1_mpls_tag(sdmproj_port1_mpls_tag),
                                        .sdmproj_port1_l2_table(sdmproj_port1_l2_table),
                                        .sdmproj_port1_pkts_count(sdmproj_port1_pkts_count),
                                        .sdmproj_port1_pkts_count_sop(sdmproj_port1_pkts_count_sop),
                                        .sdmproj_port1_oct_count(sdmproj_port1_oct_count),
                                        .sdmproj_port1_pkts_clear(sdmproj_port1_pkts_clear),
                                        .sdmproj_port1_pkts_clear_sop(sdmproj_port1_pkts_clear_sop),
                                        .sdmproj_port1_oct_clear(sdmproj_port1_oct_clear),
                                    .sdmproj_mfifo1_pkts_count_eop(sdmproj_mfifo1_pkts_count_eop),
                                    .sdmproj_mfifo1_pkts_count_sop(sdmproj_mfifo1_pkts_count_sop),

.sdmproj_mfifo1_pkts_count_eop_clear(sdmproj_mfifo1_pkts_count_eop_clear),

.sdmproj_mfifo1_pkts_count_sop_clear(sdmproj_mfifo1_pkts_count_sop_clear),
                                    .sdmproj_ipipe1_pkts_count_eop(sdmproj_ipipe1_pkts_count_eop),
                                    .sdmproj_ipipe1_pkts_count_sop(sdmproj_ipipe1_pkts_count_sop),

.sdmproj_ipipe1_pkts_count_eop_clear(sdmproj_ipipe1_pkts_count_eop_clear),

.sdmproj_ipipe1_pkts_count_sop_clear(sdmproj_ipipe1_pkts_count_sop_clear),
                                        .sdmproj_discard1_sop_count(sdmproj_discard1_sop_count),
                                        .sdmproj_discard1_eop_count(sdmproj_discard1_eop_count),
                                        .sdmproj_discard1_sop_clear(sdmproj_discard1_sop_clear),
                                        .sdmproj_discard1_eop_clear(sdmproj_discard1_eop_clear),
                                        .sdmproj_port1_link_upcount(sdmproj_port1_link_upcount),
                                        .sdmproj_port1_link_upclear(sdmproj_port1_link_upclear),
                                        .sdmproj_port1_link_dncount(sdmproj_port1_link_dncount),
                                        .sdmproj_port1_link_dnclear(sdmproj_port1_link_dnclear),
                                        .sdmproj_mb_zbt_add_out(sdmproj_mb_zbt_add_out),
                                        .sdmproj_mb_zbt_data_out(sdmproj_mb_zbt_data_out),
                                        .sdmproj_mb_zbt_we(sdmproj_mb_zbt_we),
                                        .sdmproj_mb_zbt_valid(sdmproj_mb_zbt_valid),
                                        .sdmproj_mb_zbt_data_in(sdmproj_mb_zbt_data_in),
                                        .sdmproj_mb_zbt_rd(sdmproj_mb_zbt_rd),
                                        .sdmproj_mb_stats_we(sdmproj_mb_stats_we),
                                        .sdmproj_mb_stats_valid(sdmproj_mb_stats_valid),
                                        .sdmproj_mb_stats_data_in(sdmproj_mb_stats_data_in),
                                        .sdmproj_mb_stats_rd(sdmproj_mb_stats_rd),
                                        .sdmproj_tx_rst_l(sdmproj_tx_rst_l),
                                        .sdmproj_port0_l2_error(sdmproj_port0_l2_error),
                                        .sdmproj_port0_overflow(sdmproj_port0_overflow),
                                        .sdmproj_port0_zbt_error(sdmproj_port0_zbt_error),
                                        .sdmproj_port1_l2_error(sdmproj_port1_l2_error),
                                        .sdmproj_port1_overflow(sdmproj_port1_overflow),
                                                  .sdmproj_port1_zbt_error(sdmproj_port1_zbt_error),
                                                  .sdmproj_port0_l2_error_clear(sdmproj_port0_l2_error_clear),
                                                  .sdmproj_port0_overflow_clear(sdmproj_port0_overflow_clear),
                                                  .sdmproj_port0_zbt_error_clear(sdmproj_port0_zbt_error_clear),
                                                  .sdmproj_port1_l2_error_clear(sdmproj_port1_l2_error_clear),
                                                  .sdmproj_port1_overflow_clear(sdmproj_port1_overflow_clear),
                                                  .sdmproj_port1_zbt_error_clear(sdmproj_port1_zbt_error_clear),
                                                  .reset_port1(reset_port1),
                                                  .reset_port0(reset_port0),
                                                  .reset_mac(reset_mac)
                                                  );


/*************************************************************************\
* Link UP down counters                                                   *
\*************************************************************************/


sdmproj_link_stats usdmproj_link_stats (
                                                  .clk_SYS(clk_SYS),
                                                  .sdmproj_reset_l(sdmproj_reset_l),
                                                  .sdmproj_port0_link_state(sdmproj_port0_link_state),
                                                  .sdmproj_port1_link_state(sdmproj_port1_link_state),
                                                  .sdmproj_port0_link_upcount(sdmproj_port0_link_upcount),
                                                  .sdmproj_port0_link_upclear(sdmproj_port0_link_upclear),
                                                  .sdmproj_port0_link_dncount(sdmproj_port0_link_dncount),
                                                  .sdmproj_port0_link_dnclear(sdmproj_port0_link_dnclear),
                                                  .sdmproj_port1_link_upcount(sdmproj_port1_link_upcount),
                                                  .sdmproj_port1_link_upclear(sdmproj_port1_link_upclear),
                                                  .sdmproj_port1_link_dncount(sdmproj_port1_link_dncount),
                                                  .sdmproj_port1_link_dnclear(sdmproj_port1_link_dnclear)
                                              );



/*************************************************************************\
* This is the Sdmproj NT SRAM interface for the address lookup            *
\*************************************************************************/
wire   sdmproj_zbt_en;
assign sdmproj_zbt_en = 1'b1;

sdmproj_zbt usdmproj_zbt                         (
                                                  .clk_SYS(clk_SYS),
                                                  .clk_2xSYS(clk_2xSYS),
                                                  .sdmproj_reset_l(sdmproj_reset_l),
                                                  .mb_clk_en(mb_clk_en),
                                                  .sdmproj_zbt_en(sdmproj_zbt_en),
                                                  .sdmproj_mb_zbt_add_out(sdmproj_mb_zbt_add_out),
                                                  .sdmproj_mb_zbt_data_out(sdmproj_mb_zbt_data_out),
                                                  .sdmproj_mb_zbt_we(sdmproj_mb_zbt_we),
                                                  .sdmproj_mb_zbt_valid(sdmproj_mb_zbt_valid),
                                                  .sdmproj_mb_zbt_afull(sdmproj_mb_zbt_afull),
                                                  .sdmproj_mb_zbt_data_in(sdmproj_mb_zbt_data_in),
                                                  .sdmproj_mb_zbt_rd(sdmproj_mb_zbt_rd),
                                                  .sdmproj_p0_zbt_add_out(sdmproj_p0_zbt_add_out),
                                                  .sdmproj_p0_zbt_we(sdmproj_p0_zbt_we),
                                        .sdmproj_p0_zbt_data_out(sdmproj_p0_zbt_data_out),
                                        .sdmproj_p0_zbt_rd(sdmproj_p0_zbt_rd),
                                        .sdmproj_p0_zbt_valid(sdmproj_p0_zbt_valid),
                                        .sdmproj_p0_zbt_afull(sdmproj_p0_zbt_afull),
                                        .sdmproj_p1_zbt_add_out(sdmproj_p1_zbt_add_out),
                                        .sdmproj_p1_zbt_we(sdmproj_p1_zbt_we),
                                        .sdmproj_p1_zbt_data_out(sdmproj_p1_zbt_data_out),
                                        .sdmproj_p1_zbt_rd(sdmproj_p1_zbt_rd),
                                        .sdmproj_p1_zbt_valid(sdmproj_p1_zbt_valid),
                                        .sdmproj_p1_zbt_afull(sdmproj_p1_zbt_afull),
                                        .sdmproj_vlanram_cs(sdmproj_vlanram_cs),    //RA 19
                                        .sdmproj_vlanram_oe_l(sdmproj_vlanram_oe_l), //RA 19
                                        .sdmproj_vlanram_we_l(sdmproj_vlanram_we_l), //RA 19
                                        .sdmproj_vlanram_adv(sdmproj_vlanram_adv),  //RA 19
                                        .sdmproj_vlanram_addr(sdmproj_vlanram_addr), //RA 19
                                        .sdmproj_vlanram_data(sdmproj_vlanram_data), //RA 19
                                        .sdmproj_port0_zbt_error(sdmproj_port0_zbt_error),
                                        .sdmproj_port1_zbt_error(sdmproj_port1_zbt_error),
                                        .sdmproj_port0_zbt_error_clear(sdmproj_port0_zbt_error_clear),
                                        .sdmproj_port1_zbt_error_clear(sdmproj_port1_zbt_error_clear)
                                        );
/**************************************************************************\
* This is the Sdmproj NT SRAM interface for the address lookup               *
\**************************************************************************/
wire   sdmproj_stats_en;
assign sdmproj_stats_en = 1'b1;

sdmproj_stats usdmproj_stats            (
                                        .clk_SYS(clk_SYS),
                                        .clk_2xSYS(clk_2xSYS),
                                        .sdmproj_reset_l(sdmproj_reset_l),
                                        .mb_clk_en(mb_clk_en),
                                        .sdmproj_stats_en(sdmproj_stats_en),
                                        .sdmproj_mb_zbt_add_out(sdmproj_mb_zbt_add_out),
                                        .sdmproj_mb_zbt_data_out(sdmproj_mb_zbt_data_out),
                                        .sdmproj_mb_stats_we(sdmproj_mb_stats_we),
                                        .sdmproj_mb_stats_valid(sdmproj_mb_stats_valid),
                                        .sdmproj_mb_stats_afull(sdmproj_mb_stats_afull),
                                        .sdmproj_mb_stats_data_in(sdmproj_mb_stats_data_in),
                                        .sdmproj_mb_stats_rd(sdmproj_mb_stats_rd),
                                        .sdmproj_port0_vlan_rd(sdmproj_port0_vlan_rd),
                                        .sdmproj_port0_vlan_dout(sdmproj_port0_vlan_dout),
                                        .sdmproj_port0_vlan_valid(sdmproj_port0_vlan_valid),
                                        .sdmproj_port1_vlan_rd(sdmproj_port1_vlan_rd),
                                        .sdmproj_port1_vlan_dout(sdmproj_port1_vlan_dout),
                                        .sdmproj_port1_vlan_valid(sdmproj_port1_vlan_valid),
                                        .sdmproj_statsram_cs(sdmproj_statsram_cs),      //RA 19
                                        .sdmproj_statsram_oe_l(sdmproj_statsram_oe_l),  //RA 19
                                        .sdmproj_statsram_we_l(sdmproj_statsram_we_l),  //RA 19
                                        .sdmproj_statsram_adv(sdmproj_statsram_adv),    //RA 19
                                        .sdmproj_statsram_addr(sdmproj_statsram_addr),  //RA 19
                                        .sdmproj_statsram_data(sdmproj_statsram_data)   //RA 19
                                        );


/**************************************************************************\
* This is the Pipe 0 interface the IPIPE module contains all the data    *
* processing for the data extracted from CHANNEL, frame delimitation,     *
* label extraction, byte count. Currently the missing two ports one for   *
* the MPLS label fifo which should go to the external SRAM and one for    *
* the counter sram............                                            *

\*************************************************************************/

sdmproj_ipipe usdmproj_ipipe0        (
                                      .clk_CHAN(sdmproj_chan0_tclk_155_int),
                                      .clk_SYS(clk_SYS),
                                      .sdmproj_reset_l(sdmproj_reset_l),
                                      .sdmproj_chan_tdat_p(sdmproj_chan0_tdat_p),
                                      .sdmproj_chan_ext_rd(sdmproj_chan0_ext_rd),
                                      .sdmproj_tx_data(sdmproj_tx0_data),
                                      .sdmproj_tx_ben(sdmproj_tx0_be),
                                      .sdmproj_tx_wen(sdmproj_tx0_wen),
                                      .sdmproj_tx_wen_i(sdmproj_tx0_wen_i),
                                      .sdmproj_tx_abr(sdmproj_tx0_abr),
                                      .sdmproj_tx_eop(sdmproj_tx0_eop),
                                      .sdmproj_tx_sop(sdmproj_tx0_sop),
                                      .sdmproj_tx_pafl(sdmproj_tx0_pafl_int),
                                      .sdmproj_port_special(sdmproj_port0_special),
                                      .sdmproj_port_delay(sdmproj_port0_delay),
                          .sdmproj_port_mode(sdmproj_port0_mode),
                                      .sdmproj_port_int_en(sdmproj_port0_int_en),
                                      .sdmproj_port_sa(sdmproj_port0_sa),
                                      .sdmproj_port_mpls_tag(sdmproj_port0_mpls_tag),
                                      .sdmproj_port_l2_table(sdmproj_port0_l2_table),
                                      .sdmproj_port_pkts_count(sdmproj_port0_pkts_count),
                                      .sdmproj_port_pkts_count_sop(sdmproj_port0_pkts_count_sop),
                                      .sdmproj_port_oct_count(sdmproj_port0_oct_count),
                                      .sdmproj_port_pkts_clear(sdmproj_port0_pkts_clear),
                                      .sdmproj_port_pkts_clear_sop(sdmproj_port0_pkts_clear_sop),
                                      .sdmproj_port_oct_clear(sdmproj_port0_oct_clear),
                                      .sdmproj_mfifo_pkts_count_eop(sdmproj_mfifo0_pkts_count_eop),
                                      .sdmproj_mfifo_pkts_count_sop(sdmproj_mfifo0_pkts_count_sop),

            .sdmproj_mfifo_pkts_count_eop_clear(sdmproj_mfifo0_pkts_count_eop_clear),

            .sdmproj_mfifo_pkts_count_sop_clear(sdmproj_mfifo0_pkts_count_sop_clear),
                                      .sdmproj_ipipe_pkts_count_eop(sdmproj_ipipe0_pkts_count_eop),
                                      .sdmproj_ipipe_pkts_count_sop(sdmproj_ipipe0_pkts_count_sop),

            .sdmproj_ipipe_pkts_count_eop_clear(sdmproj_ipipe0_pkts_count_eop_clear),

            .sdmproj_ipipe_pkts_count_sop_clear(sdmproj_ipipe0_pkts_count_sop_clear),
                                      .sdmproj_discard_sop_count(sdmproj_discard0_sop_count),
                                      .sdmproj_discard_eop_count(sdmproj_discard0_eop_count),
                                      .sdmproj_discard_sop_clear(sdmproj_discard0_sop_clear),
                                      .sdmproj_discard_eop_clear(sdmproj_discard0_eop_clear),
                                      .sdmproj_p_zbt_add_out(sdmproj_p0_zbt_add_out),
                                      .sdmproj_p_zbt_we(sdmproj_p0_zbt_we),
                                      .sdmproj_p_zbt_data_out(sdmproj_p0_zbt_data_out),
                                      .sdmproj_p_zbt_rd(sdmproj_p0_zbt_rd),
                                      .sdmproj_p_zbt_valid(sdmproj_p0_zbt_valid),
                                        .sdmproj_p_zbt_afull(sdmproj_p0_zbt_afull),
                                        .sdmproj_port_l2_error(sdmproj_port0_l2_error),
                                        .sdmproj_port_overflow(sdmproj_port0_overflow),
                                        .sdmproj_port_l2_error_clear(sdmproj_port0_l2_error_clear),
                                        .sdmproj_port_overflow_clear(sdmproj_port0_overflow_clear),
                                        .sdmproj_vlan_rd(sdmproj_port0_vlan_rd),
                                        .sdmproj_vlan_dout(sdmproj_port0_vlan_dout),
                                        .sdmproj_vlan_valid(sdmproj_port0_vlan_valid),
                                        .debug_bits(debug_bits),
                                        .sdmproj_spare1(sdmproj_port0_spare1),
                                        .sdmproj_spare2(sdmproj_port0_spare2)
                                        );


/***************************************************************************\
* This is the Pipe 1 interface the IPIPE module contains all the data       *
* processing for the data extracted from CHANNET, frame delimitation,       *
* label extraction, byte count. Currently the missing two ports one for     *
* the MPLS label fifo which should go to the external SRAM and one for      *
* the counter sram............                                              *

\***************************************************************************/


sdmproj_ipipe usdmproj_ipipe1          (
                                        .clk_CHAN(sdmproj_chan1_tclk_155_int),
                                        .clk_SYS(clk_SYS),
                                        .sdmproj_reset_l(sdmproj_reset_l),
                                        .sdmproj_chan_tdat_p(sdmproj_chan1_tdat_p),
                                        .sdmproj_chan_ext_rd(sdmproj_chan1_ext_rd),
                                        .sdmproj_tx_data(sdmproj_tx1_data),
                                        .sdmproj_tx_ben(sdmproj_tx1_be),
                                        .sdmproj_tx_wen(sdmproj_tx1_wen),
                                        .sdmproj_tx_wen_i(sdmproj_tx1_wen_i),
                                        .sdmproj_tx_abr(sdmproj_tx1_abr),
                                        .sdmproj_tx_eop(sdmproj_tx1_eop),
                                        .sdmproj_tx_sop(sdmproj_tx1_sop),
                                        .sdmproj_tx_pafl(sdmproj_tx1_pafl_int),
                                        .sdmproj_port_special(sdmproj_port1_special),
                                        .sdmproj_port_delay(sdmproj_port1_delay),
                                .sdmproj_port_mode(sdmproj_port1_mode),
                                        .sdmproj_port_int_en(sdmproj_port1_int_en),
                                        .sdmproj_port_sa(sdmproj_port1_sa),
                                        .sdmproj_port_mpls_tag(sdmproj_port1_mpls_tag),
                                        .sdmproj_port_l2_table(sdmproj_port1_l2_table),
                                        .sdmproj_port_pkts_count(sdmproj_port1_pkts_count),
                                        .sdmproj_port_pkts_count_sop(sdmproj_port1_pkts_count_sop),
                                        .sdmproj_port_oct_count(sdmproj_port1_oct_count),
                                        .sdmproj_port_pkts_clear(sdmproj_port1_pkts_clear),
                                        .sdmproj_port_pkts_clear_sop(sdmproj_port1_pkts_clear_sop),
                                        .sdmproj_port_oct_clear(sdmproj_port1_oct_clear),
                                        .sdmproj_mfifo_pkts_count_eop(sdmproj_mfifo1_pkts_count_eop),
                                        .sdmproj_mfifo_pkts_count_sop(sdmproj_mfifo1_pkts_count_sop),

          .sdmproj_mfifo_pkts_count_eop_clear(sdmproj_mfifo1_pkts_count_eop_clear),
            .sdmproj_mfifo_pkts_count_sop_clear(sdmproj_mfifo1_pkts_count_sop_clear),
                                          .sdmproj_ipipe_pkts_count_eop(sdmproj_ipipe1_pkts_count_eop),
                                          .sdmproj_ipipe_pkts_count_sop(sdmproj_ipipe1_pkts_count_sop),

            .sdmproj_ipipe_pkts_count_eop_clear(sdmproj_ipipe1_pkts_count_eop_clear),

            .sdmproj_ipipe_pkts_count_sop_clear(sdmproj_ipipe1_pkts_count_sop_clear),
                                          .sdmproj_discard_sop_count(sdmproj_discard1_sop_count),
                                          .sdmproj_discard_eop_count(sdmproj_discard1_eop_count),
                                          .sdmproj_discard_sop_clear(sdmproj_discard1_sop_clear),
                                          .sdmproj_discard_eop_clear(sdmproj_discard1_eop_clear),
                                          .sdmproj_p_zbt_add_out(sdmproj_p1_zbt_add_out),
                                          .sdmproj_p_zbt_we(sdmproj_p1_zbt_we),
                                          .sdmproj_p_zbt_data_out(sdmproj_p1_zbt_data_out),
                                          .sdmproj_p_zbt_rd(sdmproj_p1_zbt_rd),
                                          .sdmproj_p_zbt_valid(sdmproj_p1_zbt_valid),
                                          .sdmproj_p_zbt_afull(sdmproj_p1_zbt_afull),
                                          .sdmproj_port_l2_error(sdmproj_port1_l2_error),
                                          .sdmproj_port_overflow(sdmproj_port1_overflow),
                                          .sdmproj_port_l2_error_clear(sdmproj_port1_l2_error_clear),
                                          .sdmproj_port_overflow_clear(sdmproj_port1_overflow_clear),
                                          .sdmproj_vlan_rd(sdmproj_port1_vlan_rd),
                                          .sdmproj_vlan_dout(sdmproj_port1_vlan_dout),
                                          .sdmproj_vlan_valid(sdmproj_port1_vlan_valid),
                                          .debug_bits(),
                                          .sdmproj_spare1(sdmproj_port1_spare1),
                                          .sdmproj_spare2(sdmproj_port1_spare2)
                                          );

/***************************************************************************\
*       This is the Sdmproj MAC interface.  This module arbitrates between *
* the two ports for time on the MAC FIFO bus. Also the internal eop, sop   *
* and abr for both ports are fed here and muxed to the outside pins        *
\***************************************************************************/


sdmproj_mac     usdmproj_mac        (
                                          .clk_SYS(clk_SYS),
                                          .clk_MAC(clk_MAC),
                                          .sdmproj_reset_l(sdmproj_reset_l),
                                          .sdmproj_mac_ctl_en(sdmproj_mac_ctl_en),
                                          .sdmproj_port0_link_state(sdmproj_port0_link_state),
                                          .sdmproj_port1_link_state(sdmproj_port1_link_state),
                                          .sdmproj_mac_p0_en(sdmproj_mac_p0_en),
                                          .sdmproj_mac_p1_en(sdmproj_mac_p1_en),
                                          .sdmproj_tx0_wen(sdmproj_tx0_wen_i),
                                          .sdmproj_tx0_sop(sdmproj_tx0_sop),
                                          .sdmproj_tx0_eop(sdmproj_tx0_eop),
                                          .sdmproj_tx0_abr(sdmproj_tx0_abr),
                                          .sdmproj_tx0_pafl_int(sdmproj_tx0_pafl_int),
                                          .sdmproj_tx1_wen(sdmproj_tx1_wen_i),
                                          .sdmproj_tx1_sop(sdmproj_tx1_sop),
                                          .sdmproj_tx1_eop(sdmproj_tx1_eop),
                                          .sdmproj_tx1_abr(sdmproj_tx1_abr),
                                          .sdmproj_tx1_pafl_int(sdmproj_tx1_pafl_int),
                    .sdmproj_tx0_ren(sdmproj_tx0_ren),
                    .sdmproj_tx0_oe_l(sdmproj_tx0_oe_l),
                    .sdmproj_tx0_efl(sdmproj_tx0_sop_efl),
                    .sdmproj_tx1_ren(sdmproj_tx1_ren),
                    .sdmproj_tx1_oe_l(sdmproj_tx1_oe_l),
                    .sdmproj_tx1_efl(sdmproj_tx1_sop_efl),
                    .sdmproj_txsel_l(sdmproj_txsel_l),
                    .sdmproj_fps_txf(sdmproj_fps_txf),
                    .sdmproj_sop_txf(sdmproj_sop_txf),
                    .sdmproj_eop_txf(sdmproj_eop_txf),
                    .sdmproj_vtg(sdmproj_vtg),
                    .sdmproj_txasis(sdmproj_txasis),
                    .sdmproj_flct(sdmproj_flct),
                    .sdmproj_flct_lat(sdmproj_flct_lat),
                    .sdmproj_txrdy_0(sdmproj_txrdy_0),
                    .sdmproj_txrdy_1(sdmproj_txrdy_1),
                    .sdmproj_flow_ctl_p0(sdmproj_flow_ctl_p0),
                    .sdmproj_flow_ctl_p1(sdmproj_flow_ctl_p1)
                );

endmodule
```

Bibliography

Baldwin, Carliss Y., and Kim B. Clark. *Design Rules, Volume 1: The Power of Modularity.* Cambridge, MA: MIT Press 2000.

Christansen, Clayton and Michael Raynor (June 1997) *Innovators Dilema: When New Technologies Cause Great Firms to Fail.* Harvard Business School Press (1997)

Chesbrough, Henry (2006) *Open Business Models* 2006 Harvard Business School Press, Boston MA

Chesbrough, Henry (2006) *Open Innovation* 2006 Harvard Business School Press, Boston MA

Smith, Gina "Unsung Innovators: Lynn Conway and Carver Mead" *Computerworld 2007*

Thompson, Clive "Build it. Share it. Profit. Can Open Source Hardware Work?" *Wired* Oct. 2008

Von Hippel, Eric. (2002) Shifting Innovation to Users via Toolkits. Management Science, Vol. 48, No.7, July 2002

Von Hippel, Eric (2005) *Democratizing Innovation.* 2005 MIT Press, Cambridge MA

Von Hippel, Eric (1986) Lead Users: A source of Novel Product Concepts. Management Science. Vol 32, No.7, July 1986

Von Hippel, Eric (1994). "Sticky Information and the locus of problem solving: Implications for innovation." Management Science, 40(4) 429-439.

Wozniak, Stephen "Homebrew and How the Apple Came to Be" _*in Steve Ditlea, ed., Digital Deli, 1984*