



DIFU 100Ci@



REVISTA DE DIFUSIÓN CIENTÍFICA

INGENIERÍA Y TECNOLOGÍAS

IPv4

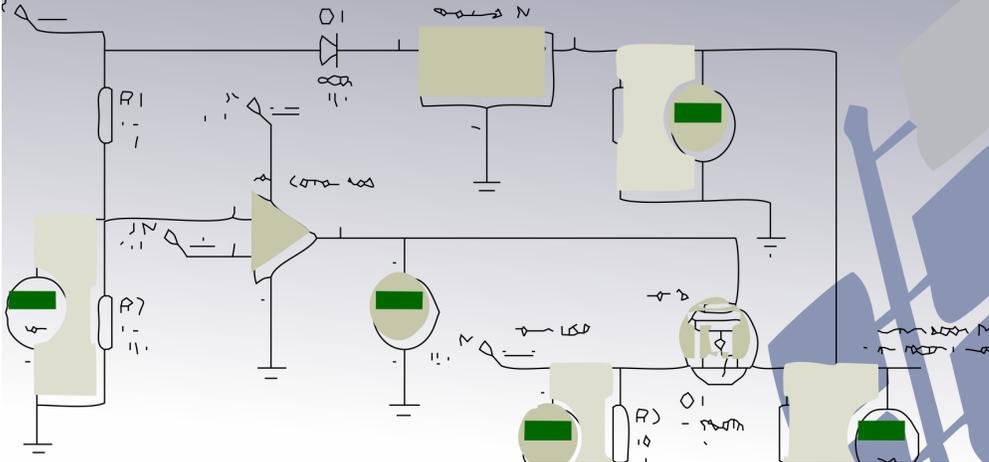


4,294,967,296
devices

IPv6



340,282,366,920,938,463,463,374,607,431,768,211,456
devices



DIFU100ci@ (léase difuciencia) Vol. 10, No.2, septiembre-diciembre 2016, es una publicación cuatrimestral editada por la Universidad Autónoma de Zacatecas, "Francisco García Salinas", a través de la Unidad Académica de Ingeniería Eléctrica. Jardín Juárez No. 147, Colonia Centro, Zacatecas, Zacatecas, C.P. 98000. Tel. (01 492) 925 6690. <http://difu100cia.uaz.edu.mx>. Editor responsable: Jorge Flores Troncoso, email: jflorest@uaz.edu.mx.

Reservas de Derechos al Uso Exclusivo No. 04-2015-060212215100-203, ISSN: 2007-3585, ambos otorgados por el Instituto Nacional de Derecho de Autor. Responsable de la última actualización de este número Jorge Flores Troncoso, López Velarde 801, Colonia Centro, Zacatecas, Zacatecas, C.P. 98000. Fecha de última actualización 16 de diciembre de 2016. Las opiniones expresadas por los autores no necesariamente reflejan la postura del editor de la publicación. Queda estrictamente prohibida la reproducción total o parcial de los contenidos e imágenes de la publicación sin previa autorización de la Universidad Autónoma de Zacatecas "Francisco García Salinas" a través de la Unidad Académica de Ingeniería Eléctrica.

Producción

**Universidad Autónoma de Zacatecas
"Francisco García Salinas"**

Diseño y Producción

Jorge Flores Troncoso

Diseño de Portada

José Ricardo Gómez Rodríguez

D.R. de la Presente Edición

**Universidad Autónoma de Zacatecas
"Francisco García Salinas"**

**López Velarde 801, Centro
98010 Zacatecas, Zac. México**

ISSN 2007-3585

DIFU100ci@ (léase difucencia) Vol. 10, No.2, septiembre-diciembre 2016, es una publicación cuatrimestral editada por la Universidad Autónoma de Zacatecas, "Francisco García Salinas", a través de la Unidad Académica de Ingeniería Eléctrica. Jardín Juárez No. 147, Colonia Centro, Zacatecas, Zacatecas, C.P. 98000. Tel. (01 492) 925 6690. <http://www.uaz.edu.mx>, <http://difu100cia.uaz.edu.mx>. Editor responsable: Jorge Flores Troncoso, email: jflorest@uaz.edu.mx.

Reservas de Derechos al Uso Exclusivo No. 04-2015-060212215100-203, ISSN: 2007-3585, ambos otorgados por el Instituto Nacional de Derecho de Autor. Responsable de la última actualización de este número Jorge Flores Troncoso, López Velarde 801, Colonia Centro, Zacatecas, Zacatecas, C.P. 98000. Fecha de última actualización 16 de diciembre de 2016.

Las opiniones expresadas por los autores no necesariamente reflejan la postura del editor de la publicación. Queda estrictamente prohibida la reproducción total o parcial de los contenidos e imágenes de la publicación sin previa autorización de la Universidad Autónoma de Zacatecas "Francisco García Salinas" a través de la Unidad Académica de Ingeniería Eléctrica.

Hecho en México
Made in Mexico

Directorio

Dr. Antonio Guzmán Fernández Rector
Dr. Rubén de Jesús Ibarra Reyes Secretario General
Dr. Luis Alejandro Aguilera Galavíz Secretario Académico
Dr. Edmundo Guerrero Sifuentes Secretario Administrativo
Dr. Manuel Reta Hernández Coord. Investigación y Posgrado
Dra. Georgia Aralu González Pérez Coord. Depto. Editorial

Editores Ejecutivos

Jorge Flores Troncoso, Editor en Jefe, UAZ, México
Manuel Reta Hernández, UAZ, México
Claudia Sifuentes Gallardo, UAZ, México
Israel Alejandro Arriaga Trejo, CONACyT-UAZ, México

Editores Asociados

Salvador Ibarra Delgado, UAZ, México
Remberto Sandoval Aréchiga, UAZ México
Juvenal Villanueva Maldonado, CONACyT-UAZ, México
Jorge Simón Rodríguez, CONACyT-UAZ, México

Grupo Revisor

Oliver Joekisch, University of Applied Sciences, Germany
Ding Hongwei, Tongji University, China
Robert Weissbach, Penn State Erie, The Behrend College
Naim Logic, Arizona Public Service
Leonardo Acho Zuppa, U Politècnica de Catalunya, España
Miguel Andrés, U. de Valencia, España
Manuel Hernández Calviño, Universidad de La Habana, Cuba
Luis C. García Santander, Universidad de Concepción, Chile
José Ángel González Fraga, UABC, México
Omar Álvarez Xochihua, UABC, México
Ramón Parra Michel, CINVESTAV, México
Leonel Soriano Equigua, FIME-UColima, México
José Luis Álvarez Flores, FIME-UColima, México
Geminiano D. Martínez Ponce, CIO, México
Jaime Sánchez García, CICESE México
Salvador Villarreal Reyes, CICESE, México
Enrique Pacheco Cabrera, AEM, México
Ernesto García Domínguez, UAZ México
Rafael Villela Varela, UAZ México
Jorge de la Torre y Ramos, UAZ México

Contenido

Editorial	p. 1
A Software Defined Interconnection Architecture for Systems on Chip.	pp. 2-6
Desarrollo de un sistema embebido de bajo costo para ?nes educativos	pp. 7-11
IPv6 El tiempo ha llegado	pp. 12-16

Editorial

La revista *DIFU*₁₀₀*ci*@ es una revista cuatrimestral que comenzó su publicación oficial en 2005. En mayo del 2012, la revista *DIFU*₁₀₀*ci*@ adquirió el ISSN.

Desde entonces, se pretende contribuir a la difusión del conocimiento de la comunidad académica tanto nacional como internacional mediante la difusión de resultados de investigación de alta calidad. La Revista se centra en obras originales, que incluyen principalmente los estudios experimentales, análisis numéricos, estudios de casos y revisiones bibliográficas que proporcionan una significativa contribución a las áreas de ingeniería y tecnología en todas las disciplinas (Electrónica, Eléctrica, Ciencias de la Computación, Mecatrónica, Robótica, Telecomunicaciones, Procesamiento de señales, Ingeniería Industrial, Ingeniería de Control, y Bioingeniería).

Desde el comienzo, la revista ha buscado la mejora de los artículos aceptados para su publicación por un proceso de evaluación por pares o árbitro de los manuscritos recibidos. Estas evaluaciones son llevadas a cabo por expertos de reconocido prestigio por sus conocimientos y logros académicos, con el objetivo de asegurar que las publicaciones seleccionadas están contribuyendo al estado del arte en diferentes áreas de interés. Además, desde su inicio, la revista se ha abierto a los estudiantes y académicos a través del Sistema Open Journal, facilitando todo el proceso de presentación y publicación.

Agradezco a los autores y revisores, que se esfuerzan para mejorar la calidad de los manuscritos. Exhorto a todos los investigadores, académicos y estudiantes en las áreas de ingeniería y tecnología para que continúen sometiendo sus artículos en nuestra revista y contribuir a la noble difusión de la ciencia y la tecnología.

Jorge Flores Troncoso
Editor en Jefe, Revista *DIFU*₁₀₀*ci*@
Universidad Autónoma de Zacatecas

A Software Defined Interconnection Architecture for Systems on Chip.

Remberto Sandoval-Arechiga, Salvador Ibarra-Delgado, Jorge Flores-Troncoso

Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica.
Av. López Velarde 801, Col. Centro, Zacatecas, Zac., México, 98000.
{rsandoval,sibarra,jflorest}@uaz.edu.mx

2016 Published by *DIFU*_{100ci}@ <http://difu100cia.uaz.edu.mx>

Abstract

On-Chip interconnections are a key component in terms of performance for Systems-on Chip (SoCs). Traditionally, buses and crossbar approaches are the designer's first choice. However, due to performance and hardware complexity limitations in complex systems, respectively bus and crossbar are replaced by routers in a Network-on-Chip approach. However, these interconnections do not adapt to requirements imposed by the SoC applications. Therefore, a flexible approach is needed. Software Defined Networking (SDN) principles bring the desired capabilities to traditional computer networks. In this paper, we propose a Software Defined Interconnection architecture based in SDN principles, and give to SoC interconnects the required flexibility and programmability in modern SoC designs.

Keywords: Interconnection, System-on-Chip, Software Defined Networking

1. Introduction

Modern digital electronic designs increase its complexity every year. Advances in semiconductors technologies have permitted tens of processing cores in a single die, allowing us to enter in the Many-Core era for Systems-on-Chip (SoCs) [4, 2].

Interconnection adaptability to communication patterns among cores in Many-Core SoCs is crucial to improve concurrent or parallel algorithm's performance [1]. However, multi-objective optimization problems overwhelm the interconnection operation process: task scheduling, task mapping, routing, power management, etc.¹[11, 18, 10, 6, 14, 15, 16]. Many-Core system

architects must build from the ground the support subsystems for every new application, leading to poor engineering process: low hardware & software reutilization, which contribute to large validation processes and recurrent costs.

In this paper we present a Software Defined Interconnection (SDI) architecture for a Many-Core SoC. With Software Defined Networking (SDN) principles as a basis we propose an architecture to orchestrate the complex multi-objective optimization process to adjust the on-chip interconnection to several applications, each with different requirements. The proposed architecture has well-defined abstraction layers and interfaces.

¹E.g. decrease power dissipation, balance traffic loads and reduce

interconnection latency.

This structure allows its deployment in parallel or in a modular fashion reducing the recurring engineering costs. Even when the work presented here is still in progress, we believe that it has the potential to become a backbone architecture of future Many-Core SoCs.

The rest of the paper is structured as follows. In section two we present the related work. In section three the SDN basic concepts are introduced. Section four presents the proposed architecture. Finally conclusions and future work are presented in section five.

2. Related Work

SDN concepts [12, 13] have become a revolution in computer networks. Its main feature is the simplification of the network management process. Although the SDN concepts have been introduced in Networks on Chip (NoC), literature does not consider a complete SDN stack architecture. Junhui Wang et al. in 2014 [17], introduced the SDN concepts for a photonic Network-on-Chip. Their work concentrates in the control and forwarding planes were optical interfaces are programmed by software via a controller. In other hand, Liu Cong et al. in 2014 [3], present a configurable, programmable and Software Defined Network-on-Chip (SDNoC). Their paper presents a router SDN architecture where the NoC routers separate the control and forwarding planes but contrary to SDN philosophy both planes are placed inside the router.

Although management architectures have not used the SDN terminology, they have been presented in the literature. In the following a brief survey is presented. An agent-based distributed resource management approach for Many-Core NoC systems is exposed in [7]. A survey including on-the-fly mapping for Multi/Many-Core NoC systems is discussed in [15]. A dynamic power and thermal management for NoC-based Many-Core systems is showed in [8]. An invasive NoC takes into account the status of the network and dynamically spawn its computation to neighboring processors is presented in [5]. Despite of the different management approaches present in the literature, it does not exist a layer structure that permits the interaction among them or a clear separation of network problems at the layers. Our work is centered in a complete SDN architecture which includes the application, network management, control and forwarding planes. In addition, we proposed the data processing plane, which include the Processing Elements (PEs).

The principal contribution of the paper is the SDI architecture which optimization (e. g. power & energy or performance) can be adjusted in a modular fashion for every new application without changing the software or hardware layers laying beneath.

3. Software Defined Networking

From the point of view of the network administrator, a simple network architecture can be divided in three planes: data forwarding, control and management. Data forwarding corresponds to the network devices (routers, switches, hubs, etc.) that transmits data from the source to its destination. The control plane corresponds to the forwarding decisions² and protocols associated to the information exchange for the decision process. Management plane includes the software services used to remotely monitor and configure the control functionality.

In computer networks the term Software Defined Networking (SDN) refers to a network architecture where the control plane is extracted from the forwarding devices. However, the network industry often has referred anything that involves software as being SDN. In this paper we follow the approach taken by [9], where SDN is defined as an architecture with four pillars:

1. The control and data forwarding planes are decoupled. Control functionality is removed from network devices that will become simple forwarding elements.
2. Forwarding decisions are flow based instead of destination based. A flow is defined by a set of packets which fields values match with some predefined criteria³.
3. Control logic is moved to an external entity, a SDN controller or a Network Operating System (NOS). A NOS generates a consistent and centralized view of the network, which applications can work with.
4. The network achieves its programmability through software applications running on top of the NOS that interacts with the underlying data plane devices.

With the aforementioned principles, we propose a *Software Defined Interconnection* (SDI) Architecture. However, it is worth noting that on-Chip interconnections have different requirements from computer networks,

²Traditionally implemented on routing tables in the data plane elements.

³E.g. a source-destination pair.

e.g. power, throughput and delay only for mention some of them. But still they share some basic principles as switching, routing, arbitration, etc. We believe that a SDI architecture will accelerate the design, implementation, performance, management and reconfiguration of SoCs through well-defined services and interfaces between abstraction layers.

4. SDI Architecture

As we pointed before a SDN architecture is centered in the administration and operation process of the network. In a Many-Core interconnection these processes will become of great importance as the power and performance requirements became harder. The proposed SDI architecture presents three layers, Operating System, Network Operating System and Infrastructure; and five planes: Applications, Network Management, Control, Data Forwarding and Data Processing, as shown in figure 1. The main difference between our architecture and computer SDN is that we included a data processing plane. In the following we describe every layer and plane in a top-down manner.

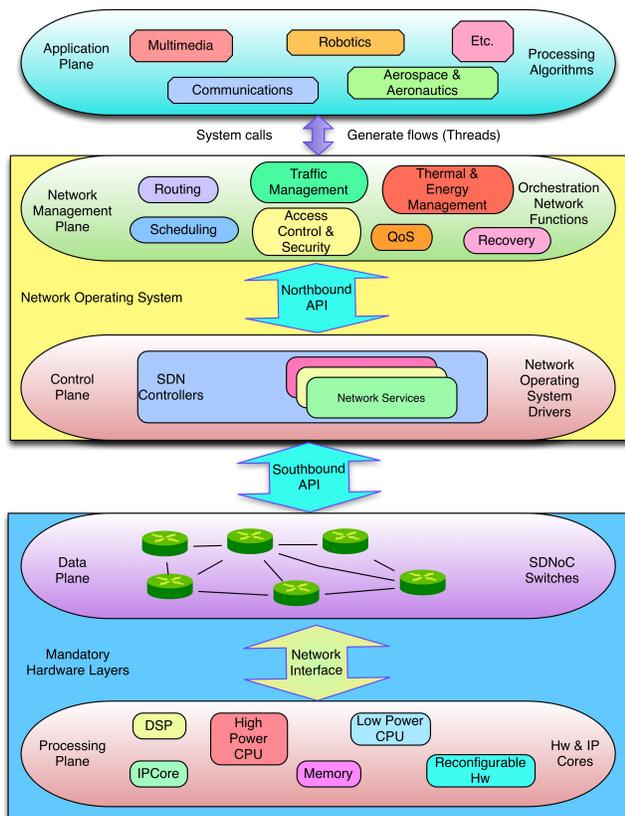


Figure 1. Software Defined Network-on-Chip Architecture.

4.1. Operating System Layer

This layer generally resides in a CPU (as a PE in the SoC) where the OS is allocated, which can be as small as a single application running on firmware, depending on what the applications' requirements impose. This layer is responsible for the interaction of the applications and the NOS. Besides, it manages the system's resource allocation such as memory, CPU time, I/O, among others.

4.1.1. Applications plane

This plane is composed by the different applications running in the SoC. Each application or processing algorithm presents a communication graph where the nodes are the functions of the PEs in the SoC interconnection, and the edges are the data dependencies among them. The weights of the edges in the graph are the average Packet Injection Rates (PIRs) for the traffic between two nodes. Then, every applications generates its own graph and requirements and present them to the NOS. The NOS will orchestrate the network functions to guarantee the requirements of the application and the operational budget of the SoC, e.g. power and thermal restrictions.

4.2. Network Operating System Layer.

The NOS layer translates the requirements given by the application layer into SoC interconnection policies and configurations that are passed to the infrastructure layer where they are executed. The NOS is composed of two planes: Control and Network Management.

4.2.1. Network Management plane

The network management plane consists of several optimization engines that modify the network functionality such as: routing, scheduling, traffic management, access control & security, QoS, thermal & energy management, recovery, etc. Such engines, called network functions, rely on a partial or global view of the interconnection network which is proportioned by control plane. Such functions allow to program the different applications running in the SoC while they maintain several QoS requirements in terms of throughput, delay and power, among others. The network management plane orchestrates the required combination of the optimization engines and organizes the execution of applications in the network. Then, network management plane generates policies and configurations that are passed to the

control plane via the northbound interface for its implementation in the data forwarding plane. The optimization engines are modular, then, if a feature is not present or has to be added/removed, the change has no impact on the others engines. This plane interacts with the applications through network operating system's calls that generate threads or flows in the network. This plane offers the following services: Optimization of an application's mapping in terms of the engines available, e.g. energy or QoS; orchestration of the network functions to optimize the on-Chip interconnection in a global manner; orchestration of the applications running in the On-Chip interconnection; mapping applications to the network (in conjunction with the NOS's calls); dynamic or static resource allocation in the on-Chip interconnection and network abstraction generation.

4.2.2. Control plane

The control plane determines the mechanisms which allow the re/configuration of the data forwarding devices. It is composed of the SDN controllers (in the SDN argot) which establish a packet or flow-based connection with the data forwarding device, via the southbound interface. Once a connection is made they can send the configuration and collect the state and statistics data from the forwarding devices. An on-Chip interconnection can have one or several SDN Controllers depending on the network size, traffic and complexity of the applications. Each SDN Controller in the on-Chip interconnection can have a partial network view, but once combined the control plane can produce a global network view. The SDN controllers may be specialized by the forwarding device they control: router or switch-based; the switching mode such as circuits or packets; etc. Then, for each kind of SDN controller a driver may be added in NOS implementations. The control plane communicates with the upper plane using the northbound interface (SDN argot). In addition the control plane offers the following services: Send configuration to a specific set of nodes in the network; collect state and statistics' data from a specific set of nodes in the network and generate a global or partial view (state) of the network.

4.3. Infrastructure Layer

The infrastructure layer is responsible for the forwarding and processing functions, it covers data forwarding and data processing planes. Due to its functions it is a hardware mandatory layer. The implementation can vary from vendors, application, customer, or budget. The only indispensable requirement for this layer is the

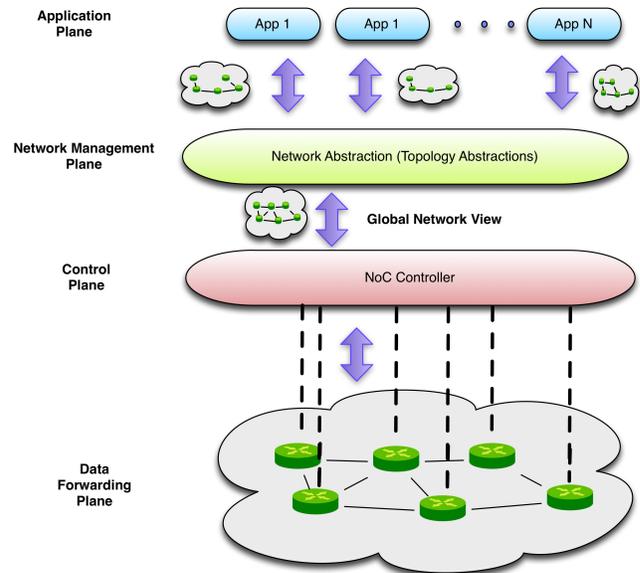


Figure 2. Global Network view and Network abstraction as services for the Applications layer.

southbound interface to communicate with the NOS layer.

4.3.1. Data Forwarding plane:

It consists of routers, Network Interfaces (NI), switches and buses that compose the interconnection in the on-Chip network. Its main function is the transmission of data from its source to destination. In this plane the interface with the upper layer is called the southbound interface in the SDN argot. The services offered by this plane are: Send data from source to destination in a flow based manner (SDN mode); send data from source to destination in a packet based manner (normal mode); execute re/configuration in the forwarding devices (for each link or router) and collect the state and statistics of the forwarding devices (for each link or router).

4.3.2. Data processing plane:

It consists of the Processing Elements (PEs) such as CPUs, IP cores, Reconfigurable Hardware, Memories, Direct Memory Access (DMA), etcetera. They are connected through NIs to the on-Chip network. Its main function is the specialized data processing specified by the applications. In this plane the interface with the upper layer is the Network Interface. The services offered by this plane are: General data processing (on CPUs); storage of data for further processing (Memories); specialized data processing (on IP cores or reconfigurable Hardware) and improve data access latency (through

DMAs). The list of services presented here for all the layers are by no means complete, as the architecture matures more services can be added, replaced or eliminated. It is of great importance to note that this architecture is from the network management point of view, i.e. only the control and administration of the NoC is considered, the data communications can follow an OSI reference model⁴

5. Conclusions and Future Work

This article presents a Software Defined Interconnection architecture for Systems-on-Chip. This proposal has the potential to overcome the interconnection management problems in a System-on-Chip design. Through well defined interfaces and abstraction layers, modifications to a layer do not impact on others. This feature improves the design and upgrade processes. In addition, Global view of the interconnection can be used to adjust its parameters to the applications running on SoCs. Resource and energy consumption of this proposal will be presented in future works.

References

- [1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, and S. W. Williams. The landscape of parallel computing research: A view from Berkeley. Technical report, Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.
- [2] S. Borkar. Thousand core chips: a technology perspective. In *Proceedings of the 44th annual Design Automation Conference*, pages 746–749. ACM, 2007.
- [3] L. Cong, W. Wen, and W. Zhiying. A configurable, programmable and software-defined network on chip. In *Advanced Research and Technology in Industry Applications (WARTIA), 2014 IEEE Workshop on*, pages 813–816. IEEE, 2014.
- [4] B. Dally. Computer architecture in the many-core era. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 1–1. IEEE, 2007.
- [5] J. Heisswolf, A. Zaib, A. Weichslgartner, M. Karle, M. Singh, T. Wild, J. Teich, A. Herkersdorf, and J. Becker. The invasive network on chip—a multi-objective many-core communication infrastructure. In *Architecture of Computing Systems (ARCS), 2014 27th International Conference on*, pages 1–8. VDE, 2014.
- [6] W. Huang, M. R. Stant, K. Sankaranarayanan, R. J. Ribando, and K. Skadron. Many-core design from a thermal perspective. In *Proceedings of the 45th annual Design Automation Conference*, pages 746–749. ACM, 2008.
- [7] S. Kobbe, L. Bauer, D. Lohmann, W. Schröder-Preikschat, and J. Henkel. Distrm: distributed resource management for on-chip many-core systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 119–128. ACM, 2011.
- [8] G. Kornaros and D. Pnevmatikatos. Dynamic power and thermal management of noc-based heterogeneous mpsoCs. *ACM Trans. Reconfigurable Technol. Syst.*, 7(1):1:1–1:26, 2 2014.
- [9] D. Kreutz, F. M. V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 1 2015.
- [10] J. L. Manferdelli, N. K. Govindaraju, and C. Crall. Challenges and opportunities in many-core computing. *Proceedings of the IEEE*, 96(5):808–815, 2008.
- [11] R. Marculescu, U. Y. Ogras, L.-S. . S. . S. Peh, N. E. Jerger, and Y. Hoskote. Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(1):3–21, 2009.
- [12] N. McKeown. Software-defined networking. In *INFOCOM 2009 Keynote talk*, volume 17, pages 30–32, 2009.
- [13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.
- [14] G. P. Nychis, C. Fallin, T. Moscibroda, O. Mutlu, and S. Seshan. On-chip networks from a networking perspective: Congestion and scalability in many-core interconnects. *ACM SIGCOMM computer communication review*, 42(4):407–418, 2012.
- [15] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, page 1. ACM, 2013.
- [16] P. Tendulkar, P. Poplavko, I. Galanommatis, and O. Maler. Many-core scheduling of data parallel applications using smt solvers. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, pages 615–622. IEEE, 2014.
- [17] J. Wang, M. Zhu, C. Peng, L. Zhou, Y. Qian, and W. Dou. Software-defined photonic network-on-chip. In *e-Technologies and Networks for Development (ICeND), 2014 Third International Conference on*, pages 127–130. IEEE, 2014.
- [18] D. H. Woo and H.-H. S. . H. S. . H. S. . H. S. Lee. Extending amdahl's law for energy-efficient computing in the many-core era. *Computer*, (12):24–31, 2008.

⁴Not exposed here to avoid confusing the reader and for space issues.

Desarrollo de un sistema embebido de bajo costo para fines educativos

K. A. Pichardo Rivas^a, I. I. Fernández Morales^a, I. A. Arriaga Trejo^b, J. Flores Troncoso^a, S. Ibarra Delgado^a, R. Sandoval Aréchiga^a, J. Villanueva Maldonado^b, J. R. Gomez-Rodriguez^a

^aCentro de Investigación y Desarrollo en Telecomunicaciones Espaciales (CIDTE), Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica.

Av. López Velarde No. 801, Zacatecas, Zac, 98000, México.

<http://www.uaz.edu.mx/>

^bCátedra CONACyT, CIDTE, Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica.

Av. López Velarde No. 801, Zacatecas, Zac, 98000, México.

<http://cidte.uaz.edu.mx/>

2017 Published by DIFU_{100ci}@ <http://difu100cia.uaz.edu.mx>

Resumen

En el documento presente se aborda la construcción de un sistema embebido de bajo costo, con dimensiones de 8cm de ancho por 7.5cm de longitud. El sistema propuesto utiliza un microcontrolador ATMEGA328P, con 23 líneas de entrada/salida de propósito general, admite programas por medio de ICSP y USB serial, siendo compatible con el software Arduino IDE mientras se tenga en la memoria del microcontrolador el bootloader. Se utilizó un programador USBASP y el programador USB serial uUSB-PA5 para cargarle los programas. Además, presenta una etapa de control y regulación de energía que le permite alimentarse por medio de una entrada USB o barrel Jack, siendo posible conectar ambas fuentes de manera simultánea sin conflicto con la suma de corrientes.

Palabras clave: Sistema embebido, microcontrolador, programación.

1. Introducción

Los sistemas embebidos son cada vez más utilizados en los aparatos electrónicos debido a las diversas aplicaciones que pueden generarse con ellos. Una persona promedio cuenta con 30 o más sistemas embebidos distribuidos en todos los aparatos en su hogar. Por lo regular están hechos para realizar una sola tarea, la cual ejecuta de manera permanente, en un circuito sencillo [1].

Debido a las necesidades de aplicaciones las cuales consumen mayor energía, se ha aumentado constante-

mente la exigencia del hardware en el cual se ejecutan tales aplicaciones. Esto ha permitido el advenimiento de sistemas embebidos y hardware cada vez más robusto y reducido en masa y volumen [2]. Los sistemas embebidos son utilizados por universidades debido a que con ellos se implementan procesos o aplicaciones específicas para monitoreo y manipulación de señales, promoviendo también el desarrollo de aplicaciones con fines educativos.

Un sistema embebido encapsula en un dispositivo todo el hardware y software requerido para implementar un sistema con propósitos específicos. Están orientados

a resolver problemas en tiempo real y cuentan con los recursos necesarios para realizar su objetivo [3]. Los sistemas embebidos trabajan con un firmware, que es el conjunto de instrucciones de programa grabado en una memoria tipo no volátil (ROM, EEPROM, flash, etc.) controlando los circuitos electrónicos para el propósito específico [4].

En el caso de sistemas embebidos que usan microcontroladores, en la memoria existe un firmware que da las instrucciones al microcontrolador para que cumpla su función o tarea asignada. Para escribir el firmware se hace uso de un programador que funciona como el nexo entre la computadora y el sistema embebido, algunos ejemplos de programadores son el USBASP, USBTiny, USB serial, etc. Los programadores seriales necesitan que en el microcontrolador haya un bootloader, un código que se ejecuta al momento de reiniciar el sistema, normalmente colocado en una parte de la memoria flash que esté protegido contra el borrado al momento de reiniciar el sistema [5].

2. Descripción del sistema embebido

El diseño que se detalla en este artículo sienta las bases para desarrollos posteriores. Por ejemplo, sirve para el diseño de una computadora a bordo para picosatélites con fines educativos.

El sistema embebido a realizar tiene como objetivo contar con entradas y/o salidas analógicas, digitales y de modulación por ancho de pulso, con un diseño de bajo costo y de dimensiones reducidas, programable mediante un puerto ICSP o un convertidor USB-Serie, igualmente se busca que acepte la entrada de programas tipo HEX.

Se tomó como referencia para el desarrollo, el sistema Arduino UNO [6], debido a que cumple con los objetivos de diseño planteados. Con el fin de representar de manera general el funcionamiento del sistema embebido desarrollado, éste se dividió en tres etapas. En la Figura 1 se muestra cada una de las etapas del sistema embebido con sus principales elementos.

2.1. Desarrollo

La etapa de regulación de energía se encarga de suministrar de energía a todo el sistema, utiliza un conector barrel Jack, que admite una entrada de 7V a 25V, un diodo de protección que evita el daño de componentes en caso de polarización inverza en el barrel jack y un regulador de voltaje a 5V que es el encargado de alimentar la etapa del microcontrolador. Se tiene también

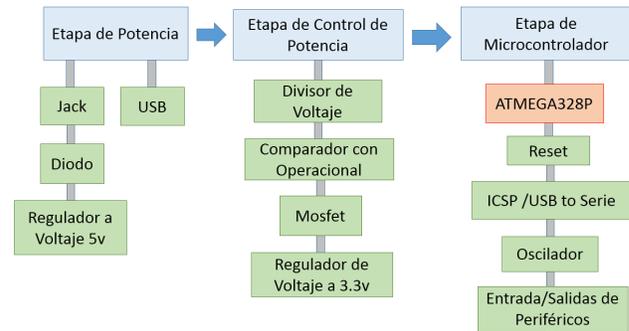


Figura 1. Etapas del funcionamiento del sistema embebido.

la opción de alimentar el sistema por medio de un USB, suministrando con 5V de manera directa.

Debido a que se pueden tener dos distintas fuentes de voltaje alimentando el sistema de manera simultánea, se generaría un problema al momento de conectarlas, la corriente generada podría llegar a dañar el sistema o a alguna de las fuentes de alimentación. La etapa de control de energía es la encargada de suprimir la corriente de la entrada USB cuando el barrel jack está conectado. Por lo tanto, podemos tener tres casos distintos:

- Cuando el barrel Jack está desconectado y el USB conectado.
- Cuando tanto el barrel Jack y el USB están conectados.
- Cuando el barrel Jack está conectado y el USB desconectado.

En el primer caso, el diodo parásito del mosfet de canal P permite el flujo de corriente de drenador a surtidor, por lo que el USB alimenta el circuito completo.

En el segundo caso, un divisor de voltaje reduce a la mitad la tensión suministrada por el barrel jack, la salida pasa hacia un amplificador operacional que actúa como comparador de voltaje, y debido a que el nivel de voltaje será mayor a 3.3V, en la salida del comparador habrán 5V, por lo tanto el mosfet se polarizará e impedirá el paso de corriente de surtidor a drenador, protegiendo así el puerto USB de la corriente de salida del regulador de 5V. Por otro lado la corriente que proviene del USB no pasará a través del mosfet debido a que para que el diodo parásito permita conducción el voltaje en el drenador tiene que ser mayor al del surtidor, dejando así, ambas fuentes completamente aisladas.

En el tercer caso el barrel Jack es el que alimenta el circuito completo, el diodo del mosfet evita el paso de corriente hacia el USB evitando que se pueda dañar el puerto.

La etapa del microcontrolador se encarga del funcionamiento de la tarea determinada, almacenada en la parte de la memoria flash del microcontrolador, donde se decidió utilizar el ATMEGA328P para que el sistema fuera compatible con el software Arduino IDE. Existe un botón de reset que tiene la función de reiniciar el programa almacenado. Se tienen dos maneras de escribir un programa en el microcontrolador: la primera es por medio de los pines ICSP (In Circuit Serial Programming), mientras que la segunda es a través de los pines Rx y Tx, haciendo uso de un convertidor USB-Serial. Para poder usar el Arduino IDE es necesario que el microcontrolador tenga el bootloader de Arduino en la memoria. En caso de que se suba algún otro programa por medio del ISCP, el bootloader de Arduino será borrado y será necesario realizar el proceso de escritura del bootloader. El circuito contiene un oscilador de cristal de 16MHz para que los pulsos generados por el microcontrolador sean más estables y fiables, en algunas aplicaciones se trabaja a grandes frecuencias o la precisión que se ocupa es mayor.

Se realizó una simulación de la etapa de regulación de energía y control de energía para probar que funciona correctamente, verificando que las dos fuentes de alimentación trabajan correctamente y no hay conflicto al momento de estar ambas activas. En la Figura 2 se muestra el comportamiento de la etapa de regulación y control de energía cuando el barrel Jack y el puerto USB están conectados, observándose que el diodo parásito del mosfet evita el paso de corriente del barrel Jack hacia el puerto USB.

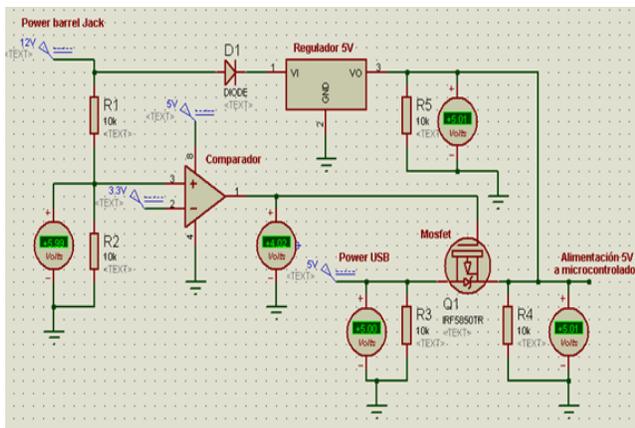


Figura 2. Etapa de regulación y control de energía con la fuente de alimentación del barrel Jack y entrada USB conectada.

En la Figura 3 se muestra el comportamiento de la etapa de regulación y control de energía cuando el puerto USB está conectado, siendo alimentado todo el sistema por el puerto USB.

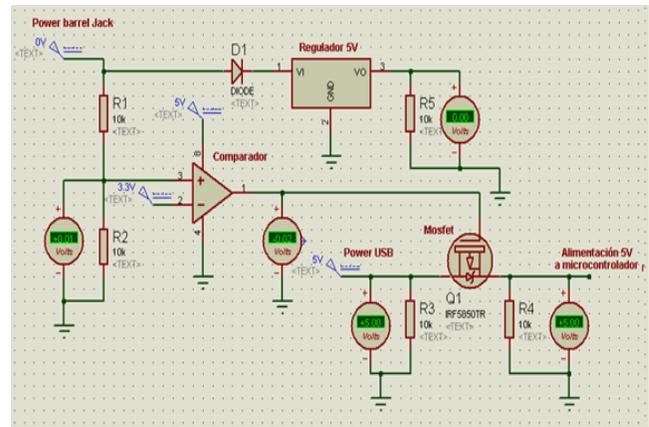


Figura 3. Etapa de regulación y control de energía con la fuente de alimentación USB conectada.

Posterior a la simulación se realizó el esquemático de todo el sistema con el software Kicad. El diagrama electrónico de la etapa de regulación y control de energía se muestra en la Figura 4, la etapa del microcontrolador se muestra en la Figura 5.

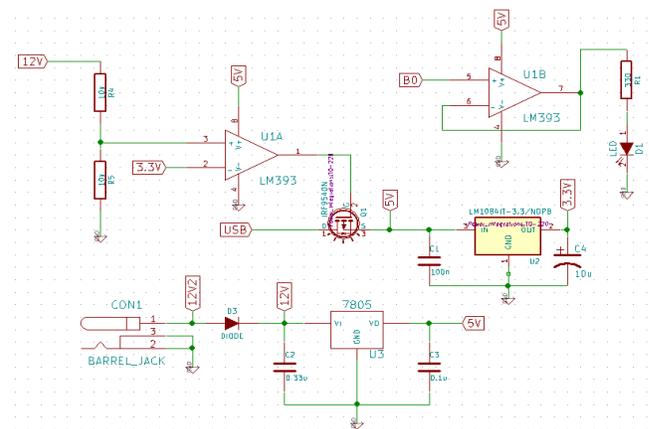


Figura 4. Esquema en Kicad de la etapa de regulación y control de energía del sistema embebido.

El diseño final de la placa se muestra en la Figura 6. Se colocó una tira de pines con distintas salidas de voltaje en caso de que se desee utilizar para alimentar algún componente, la tira tiene salida de GND, 3.3V, 5V y el voltaje de entrada que se utilice en el barrel Jack en caso de que esté conectado.

Se usó un microcontrolador ATMEGA328P sin bootloader en el sistema embebido y haciendo uso del programador USBASP se le cargó un programa con el software ATMEL STUDIO 7.0 que controla el estado intermitente del led de prueba. Posteriormente se le cargó el bootloader de Arduino para poder usar el software de Arduino IDE.

Todos los microcontroladores tienen una firma (signa-

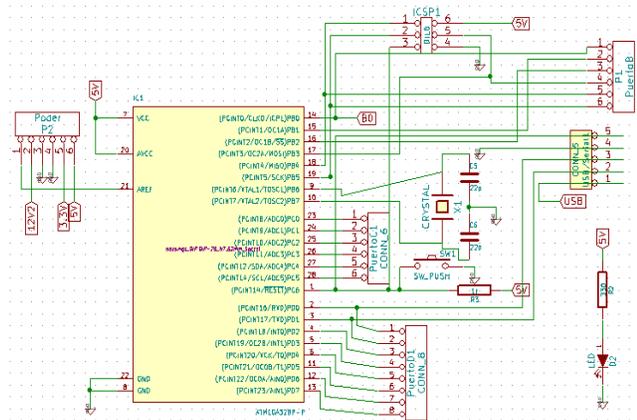


Figura 5. Esquema en Kicad de la etapa del microcontrolador del sistema embebido.

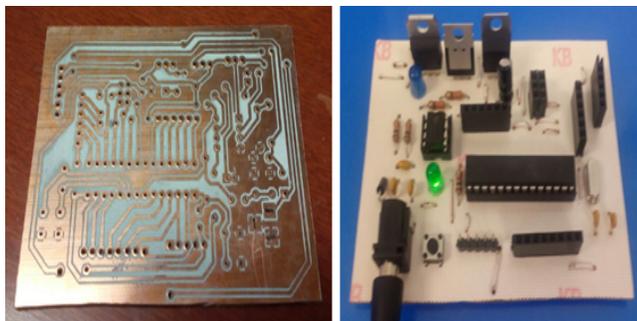


Figura 6. PCB del sistema embebido antes y después de soldar los componentes.

ture), que es un código único que los identifica a cada uno, los ATMEGA328P que se consiguen de fábrica tienen la firma 0x1e 0x95 0x14, sin embargo, los ATMEGA328P que poseen los Arduino UNO tienen la firma de 0x1e 0x95 0x0F. Para cargar el bootloader a los microcontroladores de fábrica hay que cambiar la firma que espera el Arduino IDE por la de 0x1e 0x95 0x14, para esto se tiene que modificar el archivo avrdude.conf desde la liga C:/Arduino/hardware/tools/avr/etc/avrdude.conf, una vez que se le haya cargado el bootloader, será necesario regresar el archivo avrdude.conf a su estado original para cargarle programas.

El programador USB serial que se usó fue un uUSB-PA5 [7], por medio de los pines Tx y Rx del microcontrolador se le mandó desde Arduino IDE un programa de prueba que al igual que con el USBASP controlaba el encendido y apagado de un led. A la hora de cargar el programa de esta manera es necesario presionar el botón de reset justo después de que el Arduino IDE compila el programa. La Figura 7 muestra la conexión del programador USBASP y el USB serial.

El ATMEGA328P tiene 23 pines para propósito gene-

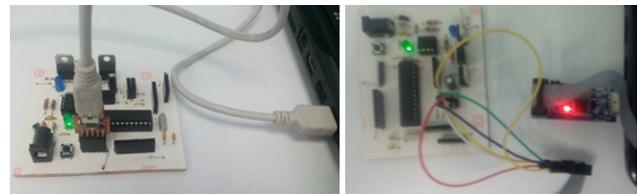


Figura 7. Sistema embebido conectado al programador USBASP y USB serial.

ral, en este caso para probar las entradas y salidas del sistema se le colocó un acelerómetro ADXL345.

3. Resultados

El sistema embebido funcionó correctamente al cargarle programas por medio del USBASP como por el USB-serial, tanto con el software ATMEL STUDIO 7.0 como con Arduino IDE, logrando controlar el encendido y apagado del led de prueba del sistema.

Para probar el funcionamiento del acelerómetro ADXL345 se usó Arduino IDE para cargarle el programa y por medio del monitor serie se observó el ángulo respecto al eje X y Y del acelerómetro. En la figura 8 se muestran los registros del acelerómetro.

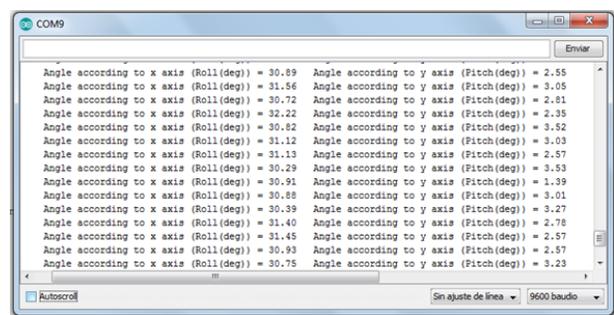


Figura 8. Ángulos respecto al eje X y Y mostrados en el monitor serie de Arduino IDE del acelerómetro conectado al sistema embebido.

3.1. Costos y materiales

Para el sistema embebido desarrollado se utilizaron los siguientes componentes mostradas en la Tabla.

Componente	Cantidad	Precio
Diodo 1N4007	1	\$2.20
Oscilador de crystal de 16MHz	1	\$13.20
Conector barrel Jack	1	\$12
Socket de 28 pines	1	\$6.60
Mosfet STP10P6F6	1	\$29.80
Regulador a 5V 7805	1	\$12.40
Regulador a 3.3V LD1117V33C	1	\$11.80
Op-amp LM358	1	\$5
Resistencias	5	\$10
Capacitores cerámicos	5	\$18
Capacitor electrolítico	1	\$5
Push buttom	1	\$2.50
Leds	2	\$6
ATMEGA328P	1	\$66
Socket de 8 pines	1	\$4
Tira de 40 pines mancho	1	\$7
Tira de 40 pines hembra	1	\$20
Placa de cobre de 10x10cm	1	\$10.50
	Total	\$ 242

Tabla 1. Lista de materiales y precios del sistema embebido desarrollado. Precios en moneda nacional mexicana.

Haciendo una comparación con otros sistemas embebidos equivalentes del mercado, se encontró que tienen un costo de aproximadamente \$440 pesos [8].

3.2. Diferencias con Arduino UNO

El sistema que se desarrolló tiene la posibilidad de ser compatible con la plataforma Arduino IDE, lo único necesario es cargarle el bootloader de Arduino desde el puerto ICSP al microcontrolador, sin embargo, si no se desea trabajar con la plataforma Arduino IDE, se puede trabajar con otras plataformas compatibles con los AVR, como la plataforma Atmel Studio o mikroBasic PRO for AVR.

La diferencia en el hardware radica principalmente en que el acabado de la tarjeta no es profesional y los componentes utilizados no son de montaje superficial, lo que abarata el costo total. La tarjeta desarrollada no incorpora un convertidor USB-Serial, como en el caso del Arduino IDE [6].

4. Conclusiones

Se realizó un sistema embebido de bajo costo y de propósito general. Este se puede programar por medio de ICSP y USB serial, tiene entradas y salidas analógicas y digitales para conectar los periféricos que se deseen. Se puede alimentar el sistema por medio de un

conector barrel Jack, una entrada USB o alguna batería que entregue 5V de corriente directa, permitiendo también usar de manera simultánea estas fuentes sin posibilidad de dañar el sistema por la suma de corrientes generada. Tiene un led que indica el encendido del sistema y un led de prueba. Se verificó que el sistema desarrollado funciona conforme a lo esperado.

Referencias

- [1] Catsoullis J., "Designing Embedded Hardware" *Oreilly*, 2005, pp. 21-50.
- [2] Vicente, C., Toledo, A., Fernández, C., & Sánchez, P., "Generación Automática de Aplicaciones Mixtas Sw/Hw mediante la Integración de Componentes COTS". *IEEE Latin America Transactions*, vol. 4, no. 2, 2006.
- [3] Sánchez Dams, Rubén Darío., Controlador lógico programable. Una mirada interna. *Editorial Universitaria de la Costa. EDUCOSTA*, ed. 1, 2009.
- [4] Sánchez Dams, Ruben Darío. "State of the art of embedded systems from an integrated perspective between hardware and software". *Revista Colombiana de Tecnologías de Avanzada*, vol. 2, no. 22, 2013.
- [5] Anton Otilia, Gelineau Brice, & Sauget Jérémy. "Firmware and bootloader". Disponible en: <https://rose.telecom-paristech.fr/2012/wp-content/uploads/2012/03/Firmwares-and-bootloaders.pdf>, 2012.
- [6] Disponible en: https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf
- [7] Disponible en: <http://www.mouser.com/ds/2/451/uUSB-PA5-Datasheet-REV1-472126.pdf>
- [8] Disponible en: <https://store.arduino.cc/product/A000066>

IPv6 El tiempo ha llegado

J.R. Gomez-Rodriguez, R. Sandoval-Aréchiga, S. Ibarra-Delgado, J. Flores-Troncoso

Universidad Autónoma de Zacatecas, Unidad Académica de Ingeniería Eléctrica.

Av. López Velarde 801, Col. Centro, Zacatecas, Zac., México, 98000.

jrgrodri@uaz.edu.mx, rem.sandoval@gmail.com, sibarra@uaz.edu.mx, jflorest@uaz.edu.mx

2016 Published by *DIFU*_{100ci}@ <http://difu100cia.uaz.edu.mx>

Resumen

En éste artículo se muestra el avance en el uso de direccionamiento ipv6 en el mundo, se muestra como a pesar de que el estandar ipv6 aprobado desde 1998 el el RFC 2460 y declarado el sucesor del estandar ipv4 (RFC791) del 1981 hasta ahora al final del 2016 se comienza a volver real la transición a uso del mismo.

Es importante mencionar el crecimiento de los dispositivos conectados a internet (Internet of Things IOT) y con ello las necesidades de direccionamiento, conoceremos como ha sido posible que ipv4 extienda su vida gracias a diferentes estrategias que lo han permitido.

Se brinda un panorama desde los Regional Internet Registry (RIR) en el estado actual del direccionamiento ipv4 y la liberacion de segmentos ipv6 a los Internet Service Provider(ISP).

Con la información que se muestra es deseable que el lector pudiera tomar desiciones y prepararse para la "Migracion Inminente".

Palabras clave: IPV6, Direccionamiento IP, Direccionamiento Público Internet.

1. Introducción

Internet se ha convertido en un recurso crítico para el funcionamiento de más y más instituciones de diversa naturaleza. Lejos están ya los días en que sólo las empresas relacionadas directamente con las tecnologías de la información eran las únicas para las cuales el acceso a Internet resultaba imprescindible para su operación. Hoy en día instituciones de toda naturaleza y tamaño requieren conectividad global ya sea para proveer servicios a través de Internet, para relacionarse con sus proveedores e incluso para el funcionamiento cotidiano de las operaciones internas. Esto implica que una interrupción en el acceso a Internet supone un alto costo, por lo que existe una fuerte demanda de meca-

nismos que brinden un alto nivel de tolerancia a fallos en la conexión a Internet. El Protocolo de Internet (IP por sus siglas en ingles) define como se comunican los dispositivos a través de las redes.[?]

Como ya es de amplio conocimiento, la encargada de administrar el direccionamiento ip es la Internet Assigned Numbers Authority (IANA). esta es responsable de la coordinación global de los sistemas de direccionamiento del Protocolo de Internet, así como de los Números de Sistemas Autónomos utilizados para enrutar el tráfico de Internet.

Actualmente hay dos tipos de direcciones IP en uso activo: IP versión 4 (IPv4) e IP versión 6 (IPv6). IPv4 se desplegó inicialmente el 1 de enero de 1983 y sigue siendo la versión más comúnmente usada. La imple-

mentación del protocolo IPv6 comenzó en 1999.

Las direcciones IPv4 e IPv6 generalmente se asignan de forma jerárquica. Los usuarios reciben direcciones IP de los proveedores de servicios de Internet (ISP). Los ISP obtienen asignaciones de direcciones IP de un Registro de Internet local (LIR) o del Registro Nacional de Internet (NIR), o de su correspondiente Registro Regional de Internet (RIR) [?]



Figura 1. Registro Regional de Internet

En ese orden de distribución para México se tiene el Network Information Center México, (NIC México) es la organización encargada de la administración del nombre de dominio territorial (ccTLD, country code Top Level Domain) .MX, el código de dos letras asignado a cada país según el ISO 3166. Entre sus funciones están el proveer los servicios de información y registro para .MX así como la asignación de direcciones de IP y el mantenimiento de las bases de datos respectivas a cada recurso.



Figura 2. Registro Nacional de Internet

Como se muestra en la figura 2 NIC México tiene subdivisiones para diversos servicios el que se encuentra en el interes del articulo es el referente mencionado como IAR México el cual como ya se ha mencionado es un Registro Nacional de Internet para recursos de Internet en México (exclusivamente). Esta facultad es otorgada mediante el contrato firmado con el Registro de Direcciones de Internet para América Latina y el Caribe (LACNIC). Las políticas utilizadas por IAR México son aquellas definidas a través de la comunidad de

Internet de América Latina en los foros públicos organizados por LACNIC, siendo las mismas para cualquier organización establecida en la región.

La ruta de asignacion de direcciones en su esquema grafico es mostrado en la figura 3.[?]

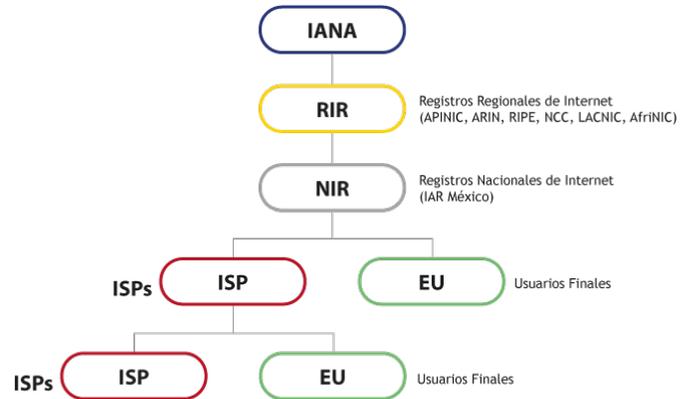


Figura 3. Esquema de Asignación de Direccionamiento

2. Protocolo de Internet versión 4 ó IPV4

El Protocolo de Internet es un protocolo de capa de red (Capa 3 tomando en referencia el modelo OSI que muestra la figura 4) diseñado en 1981 para usarse en sistemas interconectados de redes de comunicación computacional de conmutación de paquetes. El Protocolo de Internet y el Protocolo de Control de Transmisión (TCP, Transmission Control Protocol) son la base de los protocolos de Internet.

Cuenta con las siguientes características:

- Es un protocolo de un servicio de datagramas no fiable (también referido como de mejor esfuerzo).
- No proporciona garantía en la entrega de datos.
- No proporciona ni garantías sobre la corrección de los datos.
- Puede resultar en paquetes duplicados o en desorden.

2.1. Formato de Dirección IP versión 4

En una red TCP/IP a cada computadora se le asigna una dirección lógica de 32-bits que se divide en dos partes: el número de red y el número de computadora. Los 32 bits son divididos en 4 grupos de 8 bits, separados por puntos, y son representados en formato decimal.



Figura 4. Modelo de REferencia OSI

Cada bit en el octeto tiene un peso binario. El valor mínimo para un octeto es 0 y el valor máximo es 255. La siguiente figura muestra el formato básico de una dirección IP con sus 32 bits agrupados en 4 octetos. Como se muestra en ejemplo de la figura 5.

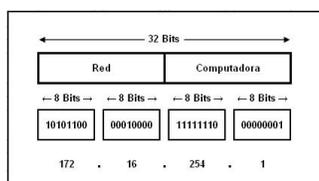


Figura 5. direccion ipv4

Este protocolo dispone de aproximadamente 4 mil millones de direcciones, otro ejemplo de la construcción del direccionamiento ipv4 se muestra en la figura 6.



Figura 6. otro ejemplo de direccion ipv4

La asignación de las direcciones ipv4 por parte de iana se muestra en la figura 7, es importante mencionar que esto ocurre cuando no se tenía un esquema ordenado ni los organismos controlaban el como se distribuirían las mismas lo que condujo a un descontrol en las asignaciones.

Al trasladarnos en el tiempo y teniendo las estructuras administrativas mencionadas al inicio del artículo se cuenta con lo necesario para administrar mejor las direcciones de esta versión ipv4, con lo que surgen mecanismos y soluciones que logran extender la utilización de esta versión por más tiempo las cuales se numeran a continuación:

1. CIDR, NAT, Direcciones privadas.

2. DHCP
3. Host Virtuales
4. Reclamar o recuperar espacios de direcciones no utilizados.
5. Control de asignación por parte de lo RIR's

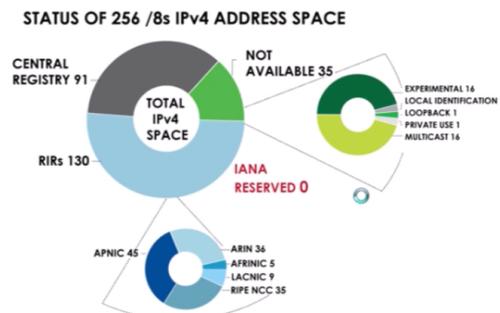


Figura 7. Distribución de direcciones IPv4

2.2. Agotamiento de direcciones ipv4

El término agotamiento IPv4 [3] se refiere a la etapa de reservas donde las asignaciones se restringen en tamaño y periodicidad. Este agotamiento implica que LACNIC, el ente encargado de asignar las direcciones para América Latina y el Caribe, no va a tener suficientes direcciones para cubrir las necesidades de direccionamiento IPv4 de sus miembros.

Con miras a solventar los problemas de agotamiento se discutieron una serie de restricciones y se definió un Manual de Políticas cuya última versión, hasta el momento de redactar este artículo, es la v2.7 del 01 de Diciembre del 2016 elaboradas por LACNIC.

Estas políticas [4] son presentadas con el propósito de asistir a los Registros de Internet en el proceso de solicitud de espacio adicional de direcciones IPv4. El factor más importante en la evaluación de las solicitudes de espacio adicional de direcciones IPv4 es la revisión del espacio actual de direcciones IPv4 de las entidades solicitantes.[?]

En la figura 8 se muestra el estado de asignación de los últimos segmentos /8 asignados a los RIR's que se tenía para el mes de octubre de 2014, en la figura 9 se muestra la misma comparación pero tomada para septiembre de 2016, los datos fueron obtenidos del sitio web Internet Number Resource Report (nro.net)

Como consecuencia de este agotamiento se pudiera prever que existirá un Mercado de direcciones IP, lo que significa la compra de IPs y cada vez va a resultar más costoso tener direccionamiento IPv4.

En Nic México se han proyectado por lo menos con 3 modelos diferentes obteniéndose fechas probables de

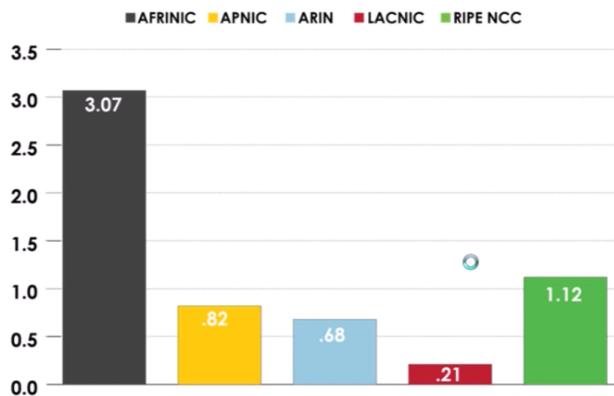


Figura 8. Agotamiento de los ultimos segmentos /8 en RIR Octubre 2014

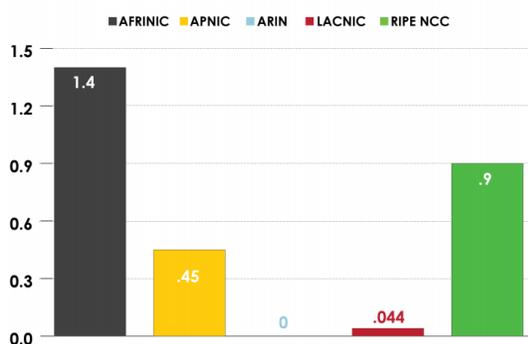


Figura 9. Agotamiento de los ultimos segmentos /8 en RIR Septiembre 2016

agotamiento de los segmentos asignados en la fase 2 del agotamiento, en los que la fecha mas lejana para el mismo es Abril de 2017. [?]

3. Protocolo de Internet versión 6 ó IPV6

Es descrito en el RFC 4291 IP Version 6 Addressing Architecture. Una dirección IPv6 tiene una longitud de 128 bits de largo y está escrita en notación hexadecimal separada por dos puntos (:).

Está compuesta por ocho números distintos, representados por 16 bits cada uno y escritos en hexadecimal, un ejemplo de una dirección IPv6 sería: **2001:0db8:9095:02e5:0216:cbff:feb2:7474.**

Las direcciones ipv6 se dividen en tres grandes porciones de dirección: el prefijo de red, el identificador de subred y un identificador del host, como se muestra en la figura 10

3.1. Alcance de las Direcciones IPV6

Cada interfaz puede tener naturalmente más de una dirección, según el alcance geográfico

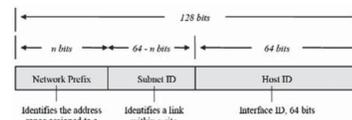


Figura 10. Dirección IPV6

Link Local: es la dirección local del interfaz con alcance de redes (LAN - Network Area Local), se puede designar o también obtenerla automáticamente componiéndose esta dirección con la dirección MAC de la interfaz, de esta manera, estas direcciones no consumen ancho de banda para relacionar la dirección IP y MAC a través del protocolo ARP como sucede en IPv4, mejorando así el desempeño de la red.

Site Local: esta dirección tiene como alcance un campus o ciudad.

Global: es la dirección que tiene cada interfaz en internet, no son modificadas como sucede en (NAT) en IPv4 facilitando así la comunicación punto a punto entre dispositivos móviles en cualquier parte del mundo.[?]

4. Diferencias entre IPV4/IPV6

Algunas de las diferencias básicas se marcan en la figura 11

DIFERENCIAS IPv4 E IPv6:		
	Pv4) Protocolo de Internet versión 4 (IPv4)	Protocolo de Internet versión 6 (IPv6)
Lanzado en	1981	1999
Tamaño de las direcciones	Número de 32 bits	Número de 128 bits
Formato de las direcciones	Notación decimal con puntos: 192.149.252.76	Notación hexadecimal: 3FFE:F200:0234:AB00:0123:4567:8901:ABCD
Notación de prefijos	192.149.0.0/24	3FFE:F200:0234::/48
Cantidad de direcciones	232 = ~4,000,000,000	2128 = ~340,000,000,000,000,000,000,000,000

Figura 11. Diferencias Basicas entre versiones de direccionamiento ip

pero existen también en el encabezado del paquete como se muestra en la figura 12

Resumiendo, las principales diferencias consideradas mejoras en el encabezado ipv6 serian:

1. Encabezado de tamaño fijo, de 40 bytes
2. Eliminacion de campos redundantes, haciendo un total de 8
3. Encabezados básicos y de extensión alineadas a un multiplo entero de 64 bits

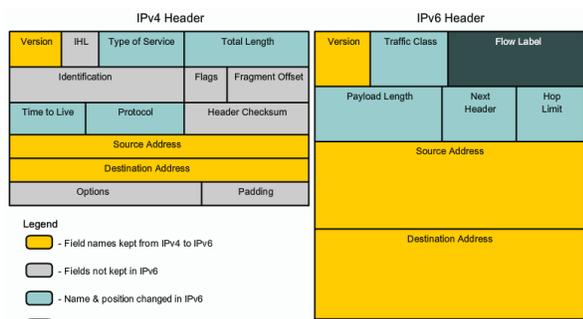


Figura 12. Diferencias Basicas entre encabezados del Paquete

4. Procesamiento eficiente de las opciones, sólo en destino y cuando éstas se presentan
5. Fragmentación procesada en el origen y el destino de los paquetes, no en los enrutadores.

5. Transición de IPV4 a IPV6

Es necesario comenzar este tema con la idea clara de que los protocolos ipv4 e ipv6 no pueden trabajar juntos esto es "no se entienden" para poder efectuar su coexistencia y así posibilitar la transición de ipv4 a ipv6 debemos conocer las tres posibles estrategias a utilizar. en orden de preferencia que se realizara así son:

1. **IPV6 Nativo** esto es usando la cabecera ipv6 desde origen hasta su destino para lo cual se tienen dos opciones:
 - Dual Stack (Doble pila)
 - Solo IPV6
2. **Tuneles** se trata de encapsular una versión de ip en otra versión.
3. **Traducción** Necesaria para comunicar dos hosts que solo hablen una versión y diferente de IP

En la figura 13,14,15 se muestran imágenes referentes a los mecanismos de transición que podrían utilizarse.

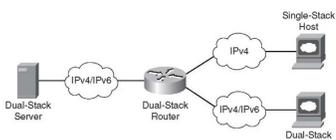


Figura 13. Mecanismo de transición a ipv6 Dual Stack

6. Conclusiones

Tomando en consideración la información aquí presentada se considera de manera importante el que se

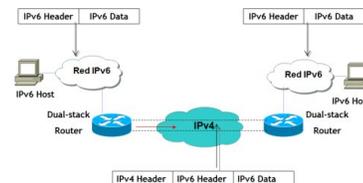


Figura 14. Mecanismo de transición a ipv6 Tuneles

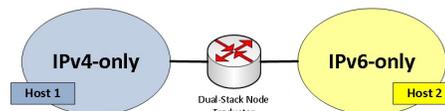


Figura 15. Mecanismo de transición a ipv6 traductor

tomen acciones a realizar en México ya que es inminente la migración de direccionamiento, esto para poder lograrla de la manera recomendada, realizando los análisis respectivos para poder seleccionar el mecanismo de transición con el que nos encontremos más cómodos.

A manera final se muestra la figura 16 obtenida de googleIPV6 sobre la adopción de ipv6 por países, el país que más avance muestra es Estados Unidos con el 30.48 % seguido de Alemania con el 27.39 %, México presenta un 0.11 %

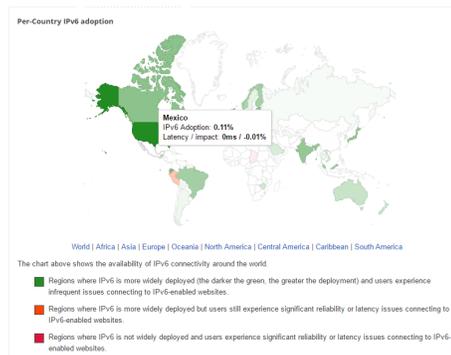


Figura 16. Porcentaje de adopción de ipv6 en México

Referencias

- [1] L. M. S. Jaimes and D. W. R. Bautista, "IPV6 en la universidad de pamplona: estado del arte". scientia et Technica, vol. 5, no. 37, pp. 415-420, 2007.
- [2] IANA, Internet Assigned Numbers Authority "Number Resources". <http://www.iana.org/numbers> 2016 Consultado Noviembre, 2016
- [3] NIC Mexico "Relación con LACNIC". <http://www.iar.mx> 2016 Consultado Noviembre, 2016
- [4] Y. Castillo, "Agotamiento IPv4 en la región latinoamericana". Prisma, vol. 5, no. 1, pp. 26-28, 2016.
- [5] NIC Mexico "Agotamiento IPV4". <http://www.lacnic.net/web/lacnic/agotamiento-ipv4> 2016 Consultado Noviembre, 2016
- [6] C. A. C. Medina and F. F. Rodríguez, "Caracterización de IPV6". Revista Tecnura, vol. 17, no. 36, pp. 111-128, 2013.

