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 $\mathcal{DIFU}_{100}ci@$ 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 Netowrking (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.

[1] E.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 were 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 decisions[2] 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 criteria[3].
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 underlaying 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,

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

[3] E.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.

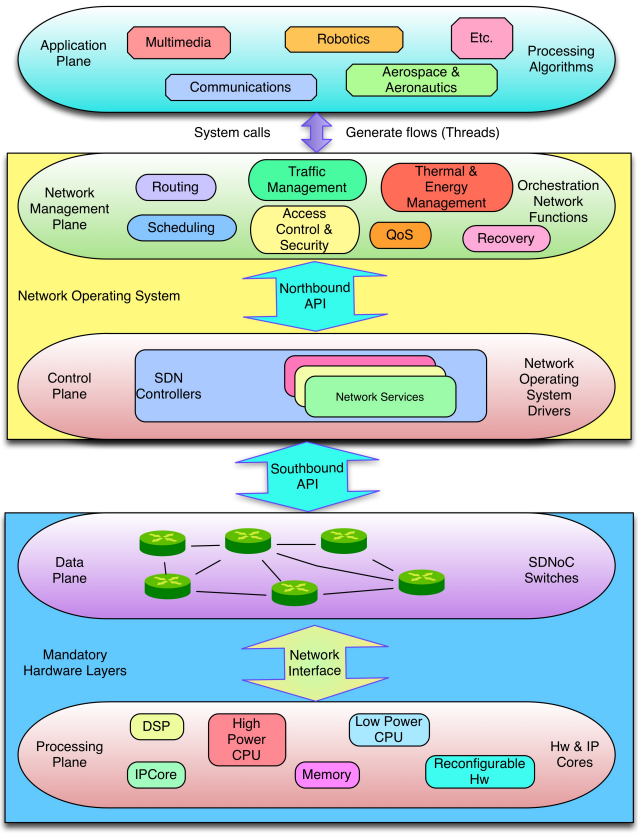

Figure 1. Software Defined Network-on-Chip Architecture.

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 fo 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

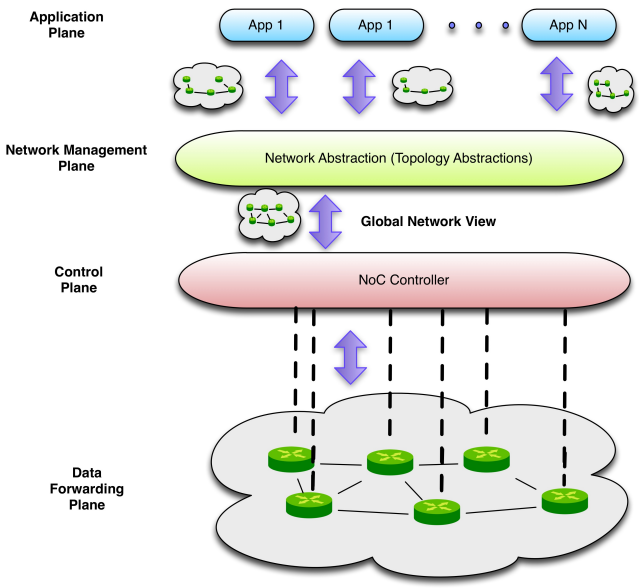

Figure 2. Global Network view and Network abstraction as services for the Applications layer.

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 model[4]

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 pf 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.

[4]Not 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]

[a]*Centro 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/
[b]*Cátedra 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 $\mathcal{DIFU}_{100}ci@$ *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 constantemente 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 asiganada. 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 programadadores 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-Serial, 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

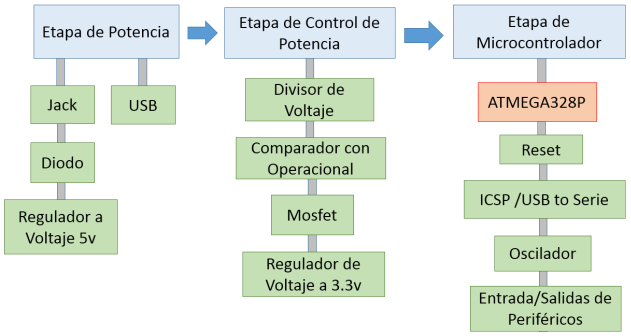

Figura 1. Etapas del funcionamiento del sistema embebido.

la opción de alimentar el sistema por medio de un USB, suminstrando 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.

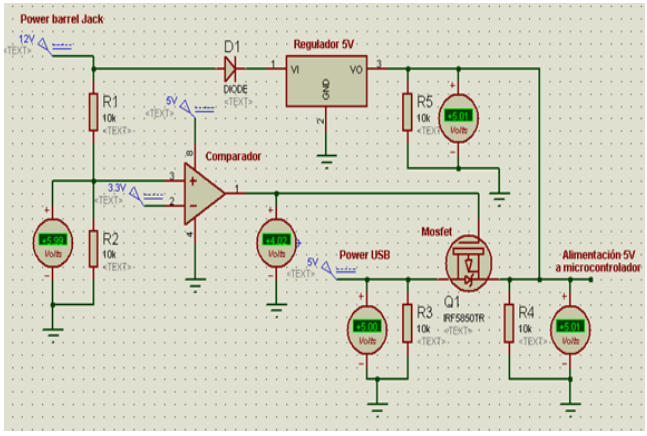

Figura 2. Etapa de regulación y control de energía con la fuente de alimentación del barrel Jack y entrada USB conectada.

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.

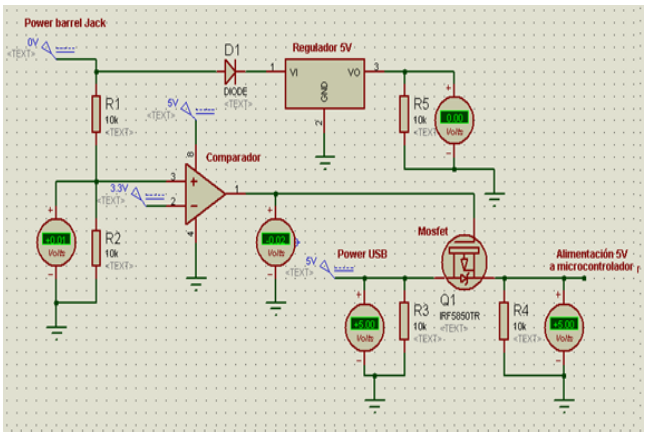

Figura 3. Etapa de regulación y control de energía con la fuente de alimentación USB conectada.

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.

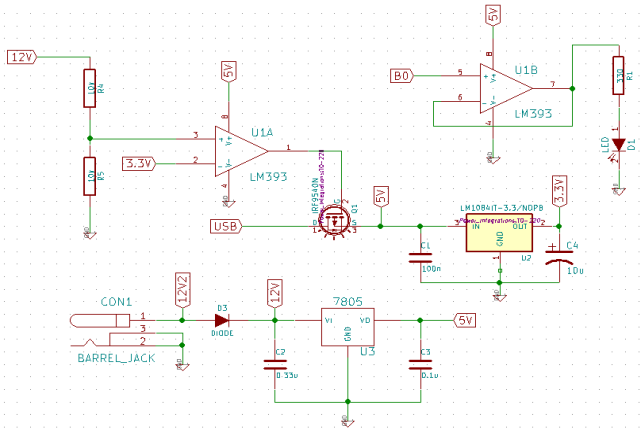

Figura 4. Esquema en Kicad de la etapa de regulación y control de energía del sistema embedido.

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 progamador 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-

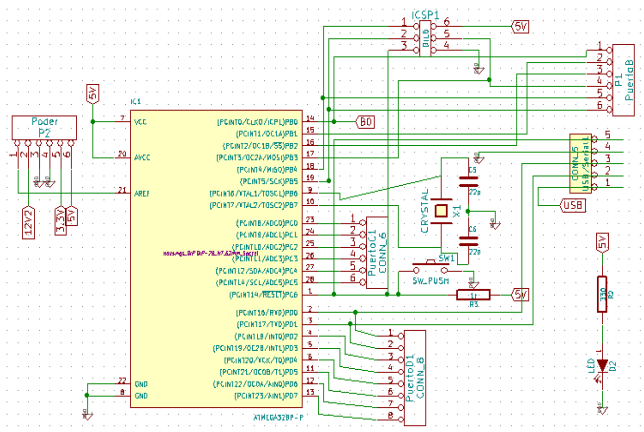

Figura 5. Esquema en Kicad de la etapa del microcontrolador del sistema embedido.

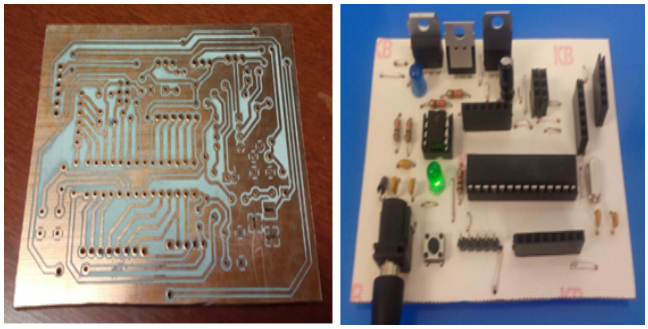

Figura 6. PCB del sistema embedido antes y después de soldar los componentes.

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-

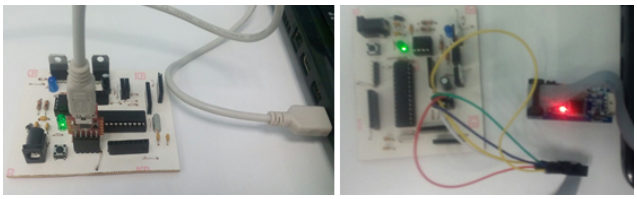

Figura 7. Sistema embebido conectado al programador USBASP y USB serial.

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.

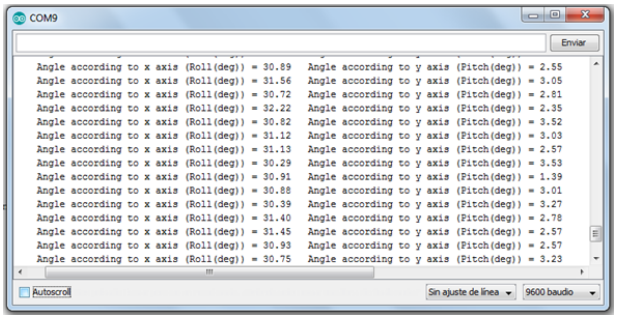

Figura 8. Ángulos respecto al eje X y Y mostrados en el monitor serie de Arduino IDE del acelerómetro conectado al sistema embebido.

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 simultanea 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