SPL 2019

IEEE

2019 X Southern Conference on Programmable Logic (SPL)

Buenos Aires - Argentina, April 10th-12th, 2019

Designer Forum

Proceedings

INTI

UNTREF

Universidad Nacional de San Luis

CAS
IEEE CIRCUITS AND SYSTEMS SOCIETY
ITCR Student Chapter

UNIVERSIDAD FASTA

utn bhi
UNIVERSIDAD TECNOLOGICA NACIONAL
Facultad Regional Bahía Blanca

AGENCIA
NACIONAL DE PROMOCION
CIENTIFICA Y TECNOLOGICA

COMISIÓN DE INVESTIGACIONES CIENTÍFICAS
Ministerio de Producción, Ciencia y Tecnología

DIEC

UNS

ISBN 978-950-658-478-8

http://www.splconf.org/spl19

Designer Forum Proceedings 2019 X Southern Conference on Programmable Logic -SPL

Cayssials, Ricardo
Designer Forum Proceedings 2019 X Southern Conference on Programmable Logic -SPL / Ricardo Cayssials ; Rodrigo Melo ; Elias Todorovich ; compilado por Ricardo Cayssials ; Rodrigo Melo ; Elias Todorovich. - 1a ed . - Tandil : Universidad Nacional del Centro de la Provincia de Buenos Aires, 2019.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-950-658-478-8

1. Informática. 2. Industria. 3. Lógica. I. Melo, Rodrigo, comp. II. Todorovich, Elias, comp. III. Título.
CDD 005.4

Editors: Cayssials, Ricardo
Melo, Rodrigo
Molina, Romina
Presso, Matías
Todorovich, Elias

X Southern Programmable Logic Conference (SPL 2019)

Designer Forum

Poster Session

Poster Session

Computadora Industrial Abierta Argentina para aplicaciones de Alta Capacidad de Cómputo

Bruno Valinoti, Rodrigo A. Melo, Noelia S. Scotti, Diego F. Alamon
Instituto Nacional de Tecnología Industrial
Centro de Micro y Nanoelectrónica
Email: {valinoti,rmelo,nscotti}@inti.gob.ar

***Resumen*—La computación de alto rendimiento fue ganando terreno tanto en el ámbito académico como en la industria, ofreciendo una solución eficiente a problemas que tiempo atrás requerían recursos excesivamente caros. Aplicaciones tales como instrumentación, multimedia o comunicaciones, requieren hoy en día de capacidades de cálculo, respuesta y manejo de interfaces cada vez más exigentes. Desde puntos de vista tanto académicos como comerciales, resulta de una importancia estratégica poseer una plataforma de desarrollo versátil, que permita encarar cualquiera de estos problemas a un costo aceptable, que sea ampliamente adoptada por la comunidad, con soporte libre y gratuito. La CIAA-ACC es una computadora industrial pensada para aplicaciones de alto rendimiento, desarrollada en Argentina, basada en dispositivos que integran un SoC doble Cortex A9 más lógica programable en un mismo circuito integrado. Además, está dotada con varias de las interfaces de comunicación más populares, tales como Gigabit Ethernet, PCI express y USB. Otra de las características que la hacen particularmente apta para este tipo de aplicaciones, son la inclusión de un conector de alta densidad de pines, del tipo FPGA Mezzanine Card, y conectores de tipo PCIe-104, que permiten realizar un arreglo de placas apiladas. En este trabajo se presenta su desarrollo, incluyendo consideraciones en dispositivos incluidos, realización del esquemático y el PCB, como así también el soporte brindado para las herramientas de Xilinx y las validaciones realizadas.**

***Index Terms*—FPGA, HPC, PCB, CIAA, Open Hardware**

I. Introducción

La Computadora Industrial Abierta Argentina (CIAA) para aplicaciones de Alta Capacidad de Cómputo (ACC) nace de una necesidad detectada por la Asociación Civil de Sistemas Embebidos (ACSE) y la Cámara Argentina de Industrias Electrónicas, Electromecánicas y Luminotécnicas (CADIEEL), con el objetivo de facilitar un salto tecnológico para las pequeñas y medianas empresas (PyMES) de la región, dedicadas al desarrollo de sistemas embebidos y la automatización de sistemas de producción.

El proyecto CIAA [1] propone una metodología de trabajo abierta y colaborativa, donde el desarrollo lo realizan instituciones académicas y empresas privadas, oficiando como nexo entre estos dos pilares de la autonomía tecnológica. El trabajo resultante se publica y libera en la modalidad de Hardware y Software Libre.

La CIAA desarrollada, presentada en este trabajo, tiene características de alto desempeño en cuánto a capacidad de cómputo e interfaces de comunicación se refiere. Está pensada para aplicaciones que requieren baja latencia en la respuesta, alta cantidad de datos procesados por unidad de tiempo, interfaces de comunicación de alta velocidad, capacidad de ejecución de un sistema operativo tipo Linux y capacidad de proceso en paralelo de tareas específicas de altas prestaciones. Los campos de aplicación de este tipo de tarjetas incluyen radares, estaciones de telecomunicación, moduladores de televisión digital, broadcasting, control vehicular, seguridad biométrica, inspección óptica en líneas de producción, realidad aumentada, control de procesos de alta confiabilidad, instrumentación médica y procesamiento de grandes volúmenes de datos, entre otros.

Este tipo de plataformas, además de tener que cumplir con las características que debieron superar el resto de los diseños CIAA, presenta grandes desafíos al momento de su diseño e implementación. Tanto desde el punto de vista de esquemáticos, implementación en hardware, montado y soldado de componentes, como también desde el del testing y soporte inicial para que la comunidad pueda utilizarla con las herramientas de desarrollo disponibles.

En este trabajo se presenta el desarrollo de una placa de circuito impreso que presenta las características necesarias para considerarse de alta capacidad de cómputo, también conocido como *High Performance Computing* (HPC), brindando a las PyMES argentinas y a la comunidad en general, una solución de alto valor agregado para incorporar en sus productos y/o proyectos de investigación. En la Fig.1 se puede apreciar un *render* de la tarjeta desarrollada con todos los componentes montados.

El trabajo está organizado de la siguiente manera: En la Sección II se explican los esquemáticos y el por qué de la selección de los componentes más relevantes. Luego en la Sección III se detallan características del PCB, la selección del *stack-up* y la simulación del diseño final con un *field solver* comercial. En las Secciones IV y V se indica cómo se dio soporte en la herramienta suministrada por el fabricante de la FPGA y los resultados de la validación con algunas pruebas sencillas, que permitieron verificar el correcto funcionamiento de los componentes y sistemas más críticos del diseño. Por último, en la Sección VI se comentan implementaciones realizadas y en proceso de la CIAA-ACC.

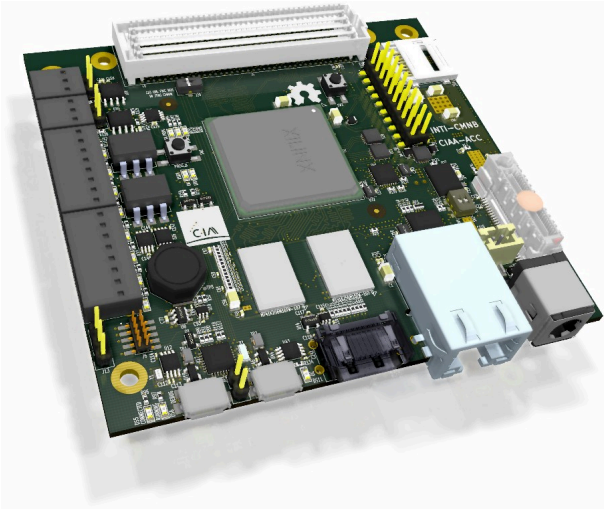


Figura 1. Render de la CIAA-ACC.

Cuadro I
CARACTERÍSTICAS PRINCIPALES DE LA TARJETA

CPU	Dual ARM Cortex A9 @ 800 MHz
FPGA	Kintex-7 (125 K celdas lógicas)
RAM	1GB DDR3
Video	HDMI Dual Role
Flash	128 Mb QSPI + Micro SD
Periféricos	Gb Ethernet, USB 2.0 OTG, USB device
	PCIe One Bank
	UART, RS485, CAN, SPI, I2C, JTAG
GPIO	16 GPIO

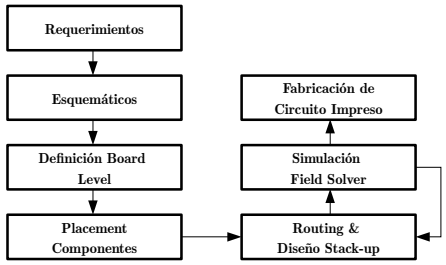


Figura 2. Flujo de diseño de PCB.

II. DEFINICIÓN DEL HARDWARE Y DISEÑO DE ESQUEMÁTICOS

II-A. Selección del hardware

Para definir la familia de microprocesadores se hizo un estudio focalizando en la potencia de cómputo, las interfaces de comunicación, los costos, la disponibilidad de documentación y la existencia de *stock* en los proveedores más conocidos. Con esta información, se seleccionó la familia de *All Programmable SoC* (APSoC) Zynq-7000 de Xilinx, en particular el dispositivo Z-7030 por tratarse de la parte con mayores prestaciones que permitiera el uso de las herramientas de diseño sin la necesidad del pago de licencias, aunque es también compatible con la Z-7045.

Una vez seleccionado el componente principal, al tratarse de una placa para HPC, surgió la necesidad de que contenga interfaces para protocolos de alta velocidad PCIe, Gigabit Ethernet, USB 2.0 y también para las normas de tipo industrial e intra placa RS485, CAN, SPI, I2C.

La fuente de alimentación se calculó para la FPGA en su máximo consumo y teniendo en cuenta las interfaces y circuitos integrados de mayor consumo. Se realizó en base a *chips* de Texas Instruments que cumple con la matriz de encendido de fuentes necesarias para la FPGA y que soporta además la carga requerida por el resto del diseño.

Debido a que CIAA-ACC fue concebida desde sus comienzos como una plataforma para HPC y aplicaciones en las cuales son muy comunes las interfaces de comunicación con una alta cantidad de pines, se decidió dotarla con un conector del tipo *FPGA Mezzanine Card* (FMC) [2] del tipo *High Pin Count* (HPC) y de un conector PCIe/104 [3], para poder utilizar un conjunto de placas apiladas, y comunicadas por medio de esta interfaz.

Las características principales de la tarjeta de desarrollo se indican en el Cuadro I.

II-B. Software EDA

Debido a las características de hardware abierto que propone el proyecto, para que cualquiera pueda utilizar los archivos de diseño, modificarlos y reproducirlos, el diseño del circuito impreso se realizó con KiCad [4], Software Libre bajo la *General Public License* (GPL), que permite realizar el flujo completo de diseño, de gran popularidad en la temática. Este software posee herramientas para diseñar esquemáticos, generar símbolos y componentes para partes no incluidas en las bibliotecas, realizar el ruteo del circuito impreso y generar los archivos para fabricación. Incluye una calculadora de líneas de transmisión, visor 3D del diseño y de gerbers, entre otras características extras.

III. PRINTED CIRCUIT BOARD (PCB)

Los circuitos digitales implementados en este tipo de diseños poseen señales de alta velocidad con tiempos de transición de estados lógicos menores al nanosegundo, y frecuencias de operación del orden de los Giga Hertz. Se debe tener especial cuidado en el *routing* de las pistas de cobre que interconectan los componentes más veloces del circuito impreso, considerando a las mismas como líneas de transmisión, con una impedancia característica y longitudes que deben mantenerse dentro de una tolerancia en todo el recorrido de la señal, para evitar la degradación de las señales. La técnica que considera estos aspectos de diseño se conoce como "Impedancia Controlada" e involucra el diseño del *stack-up*, que es el apilado de capas de la placa, el cálculo de anchos de pistas y los mecanismos de verificación posteriores a la fabricación, realizados por el usuario final o por el propio fabricante de PCBs.

El proceso de integración de un sistema complejo implica una serie de iteraciones a la hora de realizar el trazado de

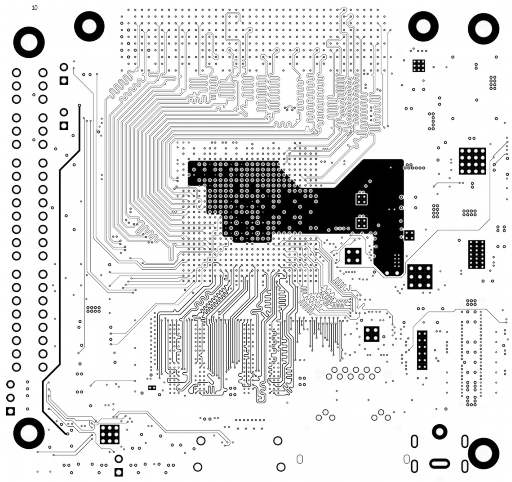
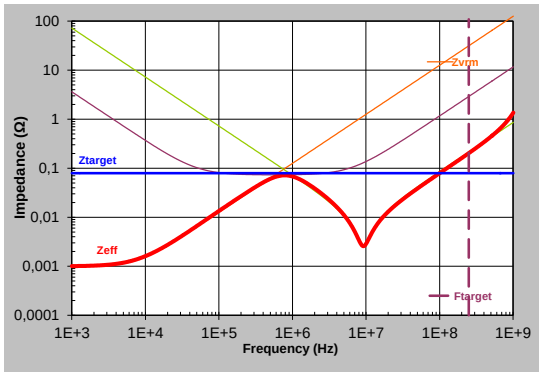


Figura 3. Capa interna de circuito impreso L2.

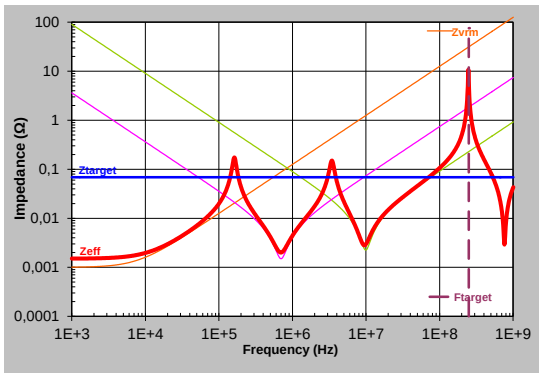
los circuitos y el conexionado de cada uno de los componentes con sus respectivas alimentaciones. En la Fig. 2 se indica brevemente un flujo de diseño y desarrollo típico donde, al principio se evalúan los requerimientos, el alcance y definiciones. Luego, en base a los mismos se realizan los esquemáticos de las diferentes partes del sistema. Una vez finalizado, se definen las características de la placa y se realiza el *placement* de los componentes, tarea donde se debe tener mucha precaución, dado que la mala ubicación de componentes puede complicar el *routing*. A partir del agregado de interfaces de alta velocidad, como las memorias DDR3, el PCIe, GEthernet, etc, para la realización del *routing*, debe definirse el *stack-up*.

En la Fig. 3 se observa uno de planos internos de circuito impreso, durante proceso de negociación del *stack-up* con el fabricante es cuando se definen los anchos de las pistas de las señales que requieran una impedancia característica de un valor determinado, ya que estas dependen de parámetros físicos que están impuestos por los materiales y procesos de fabricación utilizados. A medida que se realiza el *routing* es necesario ir verificando que el error en los parámetros buscados sea mínimo, es aquí donde resulta de gran ayuda el uso de simuladores de campo eléctrico, o *field solvers*, para conocer la interacción electromagnética entre planos de alimentación y pistas, entre pistas, etc., y poder determinar con mayor grado de seguridad los valores de impedancia y minimizar la emisión electromagnética.

Además de las consideraciones necesarias para que la degradación de las señales de alta velocidad estén dentro de las tolerancias indicadas por las normas eléctricas propias de cada protocolo, es necesario que cada uno de los componentes reciba una alimentación adecuada y estable, sin importar la exigencia de corriente o el estado y velocidad de conmutación de sus pines de entrada/salida. Esto implica que cada dominio de alimentación debe ser diseñado con rigurosidad, calculando la red de capacitores de desacople y de reservorio de energía individualmente, para mantener su impedancia por debajo de



(a) Impedancia de PDN bien diseñada



(b) Impedancia de PDN mal diseñada

Figura 4. Calculo de impedancias de dominios de alimentación.

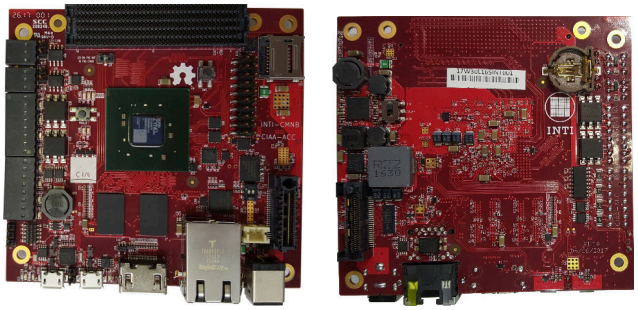
un valor límite. La distribución de la alimentación a los distintos componentes dentro una placa de circuito impreso puede buscarse en la literatura como *Power Distribution Network* (PDN). Una buena aproximación para el diseño de las PDNs puede encontrarse en [5].

De manera resumida, para el diseño de una PDN hay que cumplir con la premisa de entregar en todo momento una tensión de referencia estable y lo suficientemente libre de *glitches* y ruido, como para que los componentes asociados no incurran en un mal funcionamiento.

De acuerdo a la ley de Ohm, $V_{ripple} = I_{transitorio} * Z_{PDN}$, donde V_{ripple} es la máxima tensión de *rippple* limitante dentro del dominio de alimentación, $I_{transitorio}$ es la corriente pico de transitorio, la diferencia entre la corriente de régimen permanente y la máxima, dividido el tiempo en que se llega desde la primera hasta la segunda $I_2 - I_1/t_2 - t_1$. Z_{PDN} es la impedancia que presenta la red de alimentación y es el único parámetro que podemos manejar en nuestro diseño de manera controlada y certera.

$$Z_{target} = \frac{V_{dominio} * (\%_{ripple}/100)}{I_{max\ transitorio}} \quad (1)$$

Si bien no se desarrollarán aquí las técnicas necesarias para diseñar la Z_{PDN}, se presenta la fórmula de cálculo 1 a modo de introducir el tema. En la Fig. 4(a) se muestra la impedancia de una red de alimentación calculado de manera correcta, mientras que en 4(b) el diseño es aún insuficiente.



(a) Top (b) Bottom

Figura 5. Placa CIAA-ACC.

Los archivos necesarios para generar el proyecto se encuentran en la WEB [6] y están disponibles para que cualquiera pueda descargarlos de manera libre y gratuita, para que pueda realizarle las modificaciones que desee de acuerdo a la licencia Open Source Hardware [7].

IV. Soporte

La herramienta recomendada, provista por el fabricante del dispositivo, para trabajar con los Zynq-7000, es la denomina Vivado Suite, que permite realizar:

- Síntesis e implementación.
- Generación y transferencia de *bitstream*.
- Desarrollo de *Firmware* mediante el *Software Development Kit (SDK)*, un editor basado en eclipse que corre por debajo al compilador GNU GCC.

Para facilitar el uso de estos complejos dispositivos, que necesitan una gran cantidad de configuraciones, Vivado provee archivos listos para soportar sus placas de desarrollo, y provee mecanismos para extenderlos a otros diseños. Esto es muy importante para la rápida inicialización de un proyecto, evitando no sólo tener que buscar la parte, sino más importante, preconfigurando los parámetros necesarios del Zynq, que van desde el *timing* de las DDR hasta la conexión de cada uno de los periféricos externos.

El soporte para una nueva placa se consigue a partir de realizar los *Board Interface File* (BIF), un grupo de archivos XML bien definidos, que en conjunto conforman los *Board Definition File* (BDF) a instalar en:

<INSTALLDIR>/Vivado/<VERSION>/data/boards/board_files

Si bien el apéndice A de [8], da algunas indicaciones referentes, lo más sencillo es buscar una placa similar y modificar los archivos existentes.

El BDF de la CIAA-ACC, junto la guía de usuario, archivo de *constraints* maestro, BSP para Linux e indicaciones de como generar una imagen lista para arrancar desde una micro SD card, etc, se encuentran disponibles libremente en [9].

V. Validación

Una vez recibida la primer tanda de placas fabricadas (Fig. 5), con todos sus componentes montados y soldados, se comenzó con la verificación del funcionamiento de la fuente de alimentación. Dicho proceso implicó controlar los valores de tensión de cada uno de los dominios de alimentación, luego la matriz de encendido y por ultimo la tensión de *ripple*.

En cuanto al testing funcional, la primera prueba a ejecutar fue el famoso “Hello world!”, para verificar que el procesador y sus UARTs estuvieran funcionando. Luego se realizó la escritura/lectura de la totalidad de la memoria RAM, configurada en la máxima frecuencia que permiten los procesadores. Vale la pena resaltar que esta prueba es una de las más críticas, debido a que el ruteo de las memorias DDR3 presenta una alta complejidad, por la alta frecuencia de operación. Se pudo verificar el correcto funcionamiento de la memoria DDR3 mediante pruebas a medida y las que vienen precargadas en Vivado SDK, como así también para el resto de los periféricos y de la parte FPGA.

VI. Conclusiones

Se logró completar el diseño de un circuito impreso de alta velocidad, con doce capas eléctricas en un espacio reducido, incorporándole numerosas interfaces de comunicación. Este diseño requirió de conocimientos específicos sobre la temática “Integridad de señal” para ser llevado a cabo. Durante la etapa de diseño, se lograron ampliar los conocimientos sobre ruteo de interfaces de alta velocidad y sobre el uso de las herramientas de simulación y cálculo sobre circuitos impresos. La plataforma CIAA-ACC pudo ser utilizada en varios proyectos, de forma exitosa, hasta ahora sin mayores dificultades. Las dimensiones de la placa de circuito impreso son 90x96 mm y se alimenta con una sola fuente de 5V @ 8A.

Al momento de escribir este trabajo, además de las diez unidades existentes, se están fabricando tres nuevas unidades para *The Abdus Salam International Centre for Theoretical Physics* (ICTP) y están planificadas siete más para el *Istituto Nazionale di Fisica Nucleare* (INFN) de Italia, las cuáles serán empleadas en proyectos relacionados con detección de partículas de alta energía, en cooperación con *The European Organization for Nuclear Research* (CERN).

VII. Agradecimientos

Agradecemos a todos los colaboradores de la CIAA-ACC: Ariel Lutenberg, Pablo Ridolfi, Martín Ribelotta, así como también a las instituciones ACSE y CADIEEL.

Referencias

[1] CIAA. (2018). [Online]. Available: www.proyectociaa.com.ar
[2] *FMC, FPGA Mezzanine Card*, ANSI/VITA Std. 57.1, 2010.
[3] *PCI/104-Express & PCIe/104 Specification*, PC/104 Consortium Std. 3.0, 2015. [Online]. Available: pc104.org/wp-content/uploads/2015/03/-PCI104_Express_v3_0.pdf
[4] KiCad. (2018). [Online]. Available: kicadpcb.org
[5] Altera. (2009, May) Printed Circuit Board (PCB) Power Delivery Network (PDN) Design Methodology. [Online]. Available: www.intel.com/-content/dam/www/programmable/us/en/pdfs/literature/an/an574.pdf
[6] INTI CMNB. Repositorio Hardware de la CIAA ACC. [Online]. Available: github.com/ciaa/Hardware/tree/master/PCB/ACC/CIAA_ACC
[7] O. S. H. Association. (2018). [Online]. Available: www.oshwa.org/
[8] Xilinx. (2016, October) SystemLevel Design Entry. [Online]. Available: www.xilinx.com/support/documentation/sw_manuals/xilinx2016_3/ug895-vivadosystemleveldesignentry.pdf
[9] INTI CMNB, Emtech. Repositorio CIAA ACC support. [Online]. Available: github.com/ciaa/CIAA_ACC_Support

Implementación sobre FPGA de un generador de baud rate basado en divisor fraccional

Ariel Dalmas Di Giovanni, Emiliano H. Prato
Laboratorio Técnicas Digitales
Instituto de Investigaciones Científicas y Técnicas para la Defensa (CITEDEF)
Villa Martelli, Argentina
{adigiovanni@citedef.gob.ar, pratoemiliano@yahoo.com.ar}

***Abstract*—El trabajo presenta el desarrollo e implementación de un generador de baud rate basado en un divisor fraccional, con el objetivo de resolver un problema específico: disponer de UARTs con tasas iguales o superiores a 230400 bps, y un reloj de sistema que no es múltiplo entero de éstas velocidades. Bajo estas condiciones, el error que genera un divisor de frecuencia digital simple excede lo tolerable en una comunicación serie asíncrona. Se presenta la teoría del divisor fraccional, las consideraciones de diseño, el proceso de desarrollo, los resultados de la implementación del generador sobre una FPGA, y las ventajas de uso del divisor fraccional.**

***Index Terms*—Divisor fraccional, FPGA, baud rate, UART.**

I. Introducción

El puerto serie asincrónico fue y sigue siendo un método simple de comunicación entre dos dispositivos. Por su naturaleza asíncrona, y con el propósito de sincronizar al transmisor y al receptor se consideran dos factores: la inclusión de bits *start* y *stop* [1] , y la velocidad de comunicación, común entre extremos, estableciendo así el tiempo de duración de cada bit. Es por tanto, que se torna determinante para garantizar la comunicación que el tiempo de bit esté adecuadamente definido, el bloque que genera este tiempo es lo que se conoce como *generador de baud rate*.

Un generador simple se basa en un divisor de frecuencia, el cual divide, una frecuencia de entrada, *fin*, por un valor, *Div*, obteniendo una señal a la salida de frecuencia *fout*, o lo que es lo mismo, que el cociente entre las dos frecuencias define el valor del divisor, como se expresa en (1), debiendo ser la frecuencia de entrada superior a la frecuencia de salida deseada.

$$Div = \frac{fin}{fout} \tag{1}$$

La implementación del generador simple mediante un sistema digital consiste en un contador digital, cuyo módulo estará definido por la parte entera de *Div*. Si la relación de frecuencias no es un múltiplo entero, existirá un desvío entre la frecuencia pretendida, *fout*, y la real obtenida $fout_a$. Por tanto, se define el error relativo, *Er* (2), considerando que éste debe ser inferior al 2% para garantizar la comunicación [2].

$$Er = \frac{fout - fout_a}{fout} \tag{2}$$

Para mejorar esta situación la solución más inmediata es o bien incrementar la frecuencia de entrada al divisor o que la frecuencia de entrada sea múltiplo entero de la frecuencia de salida [3]. Ésta problemática no es nueva, y es frecuente en diseño y desarrollo de SoC, desde los tiempos en lo cuales se incorporaron las UARTs como periféricos internos. La selección del reloj principal de un sistema suele ser una tarea compleja y condicionada por múltiples restricciones propias del diseño asociadas a la aplicación y a las características inherentes de cada dispositivo [4]. De esta manera las soluciones planteadas anteriormente en muchos casos no son factibles de implementar, o complejizan al sistema. La solución a estos problemas es utilizar un divisor fraccional, como lo hacen algunas arquitecturas modernas de microcontroladores, ejemplo de esto es la arquitectura Cortex-M de ARM [5].

La motivación de este trabajo tiene que ver con la necesidad de implementar sobre una FPGA varios bloques UARTs con velocidades de comunicación altas, hasta 460800 bps, con un reloj de sistema ya predefinido en 50MHz y debiendo ser éste el único dominio de reloj en todo el sistema. Estas consideraciones, dan lugar al tema del presente trabajo: la implementación de un generador de *baud rate* basado en un divisor fraccional.

II. El divisor fraccional

A. Teoría del divisor fraccional

Según se mencionó en la sección anterior, el resultado de la relación (1), entre las frecuencias de entrada - salida del divisor, *Div*, puede no ser exacto. Si bien se puede aproximar a un valor de divisor entero, *DivEnt*, mediante trucado (3), esto acarrea un error. Se puede obtener una expresión del error relativo (4) en función del divisor exacto, *Div*, y el divisor entero, *DivEnt*, reemplazando (1) y (3) en (2); concluyendo que el error aumentará conforme la diferencia entre el divisor entero se aparte del exacto.

$$DivEnt = Truncar(Div) \tag{3}$$

$$Er = 1 - \frac{Div}{DivEnt} \tag{4}$$

Para disminuir el error será necesario considerar la diferencia, *F*, entre *Div* y *DivEnt* (5), ésta es la fracción despreciada por la operación de truncado.

$$F = Div - DivEnt \qquad F < 1 \tag{5}$$

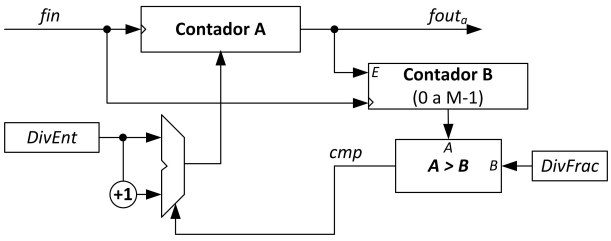


Fig. 1. Diagrama conceptual del generador.

A partir de la descripción del funcionamiento del divisor fraccional se puede comprender la importancia de considerar dicha fracción: Sea un contador digital que cuenta durante un período de tiempo desde *0 a Div-1* y en otro período de tiempo cuenta desde *0 a Div*, es decir, un pulso más que antes. Esto implica que el desborde del contador no es periódico y el tiempo promedio de desbordes dependerá de cuantos períodos cuente en un rango y cuantos períodos cuente en el otro. El parámetro que establece el período principal es derivado de la diferencia *F*, el cual es necesario cuantificarlo mediante un factor *M* (6), quedando así determinada la resolución de la parte fraccional por 2^{-M}.

$$DivFrac = 2^M F \quad M = 1, 2, 3, ... \tag{6}$$

En la Fig. 1 se muestra un esquema conceptual del divisor fraccional compuesto de dos contadores: el contador principal, Contador A, y el Contador B que determina el período del promedio definido por *M*. La salida del comparador, *cmp*, controla la cuenta del contador principal, la cual estará en estado bajo mientras la cuenta del Contador B sea menor o igual a *DivA*, y por tanto el Contador A contará un pulso más, es decir hasta un valor de *DivEnt*, en el caso contrario contará hasta *DivEnt-1*. De esta manera el valor promedio de la frecuencia de salida será más cercano al deseado. Se calcula mediante (7) el divisor aproximado, *DivA*, el cual produce la frecuencia de salida aproximada, *fout$_a$*, definida en (8).

$$DivA = DivEnt + F = DivEnt + \frac{DivFrac}{2^M} \tag{7}$$

$$fout_a = \frac{fin}{DivA} \tag{8}$$

Reemplazando (7) en (4) se obtiene la expresión del error relativo (9) en función de los parámetros fraccionales.

$$Er = 1 - \frac{Div}{DivEnt + \frac{DivFrac}{2^M}} \tag{9}$$

Del análisis de esta última expresión queda claro que el error disminuye en función que la parte fraccional se aproxime en mayor grado a la parte decimal de *Div*, lo que queda totalmente definido por el factor *M*. Cuanto mayor sea éste factor, la aproximación será cada vez más válida y por ende el error tenderá a cero.

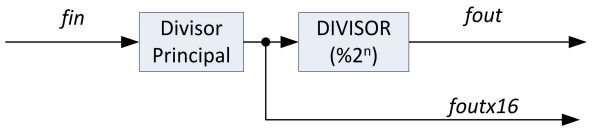


Fig. 2. Diagrama de bloques del generador de *baud rate* .

B. Generador de baud rate

Un generador de *baud rate* de UART tiene dos salidas: una para generar la base de tiempo asociada a la transmisión de los datos, de frecuencia coincidente con el *baud rate* (*fout*), y la otra salida *foutx16*, de mayor frecuencia que la anterior, utilizada por el receptor el cual debe sobremuestrear la señal de entrada para poder interpretar la información recibida. Éste factor de sobremuestreo es múltiplo de potencias de *2*. El incremento de frecuencia va en desmedro del error relativo. Para que el error esté dentro de los margenes de tolerancia apropiados es necesario derivar la salida *foutx16* de un divisor fraccional y la salida *fout* de un divisor entero, tal como se muestra en la Fig. 2.

C. Caso de estudio

La necesidad de uso de un divisor fraccional como generador de *baud rate* surge de los requerimientos para el diseño de un sistema que utiliza múltiples UARTs. Siendo el reloj del sistema de 50MHz, establecido como único dominio de reloj, y velocidades de operación de UARTs de: 230400 bps ó 460800 bps. En la Tabla. I se resumen los cálculos de los parámetros suponiendo una implementación con divisor entero. Se observa que el error relativo porcentual en ambas velocidades requeridas excede el 2%, motivo por el cual es necesario utilizar el divisor fraccional. Los cálculos presentados en la Tabla. II corresponden a una implementación con divisor fraccional. Se detallan los valores de cada divisor, la frecuencia para realizar el sobremuestreo, calculada con los valores del divisor aproximado, y el valor de la frecuencia de *baud rate*. Es notoria la disminución del error relativo, estando por debajo del 2% deseado. Se han realizado los cálculos para una velocidad de comunicación de 921600 bps, que si bien no forma parte de lo requerido, permite ver que el divisor

TABLE I
CÁLCULOS UTILIZANDO DIVISOR ENTERO

fout bps	**Div**	**DivEnt**	**foutx16 Hz**	**fout bps**	**Er%**
230400	13,563	14	3.571.428	223.214	3,12
460800	6,782	7	7.142.857	446.428	3,12
921600	3,391	3	16.666.667	1.041.666	13,03

TABLE II
CÁLCULOS UTILIZANDO DIVISOR FRACCIONAL

fout bps	**Div**	**DivEnt**	**DivFrac**	**foutx16 kHz**	**fout bps**	**Er%**
230400	13,563	13	9	3686,64	230414	-0,0064
460800	6,782	6	13	7339,45	458715	0,45
921600	3,391	3	6	14814,81	925925	-0,47

fraccional es solución como generador de *baud rate* para esa tasa. Se puede observar en los cálculos para esa condición que la división entera posee muy pocas cuentas, motivo por el cual es imprescindible considerar en el divisor la fracción, ya que la relación de frecuencia de entrada - salida al mismo no es exacta, reforzando el uso del divisor fraccional para estas tasas.

Se ha adoptado un valor de $M = 4$, ya que de esta manera se logra aproximar la fracción a un valor inferior a la décima ($2^{-4} = 0,0625$).

Por todo lo anterior, es que se optó por implementar un generador de *baud rate* basado en un divisor fraccional.

III. Implementación

A. Bloque divisor

El bloque divisor fraccional implementado, *baud_gen_frg*, posee como interfaz externa dos entradas: reloj (*clk_i*) y reset sincrónico (*reset_i*); y dos salidas: salida de frecuencia coincidente con el *baud rate* (*clkbaud_o*) y salida de frecuencia 16 veces la del *baud rate* (*clkbaudx16_o*).

El esquema de la Fig. 3 presenta tanto la interfaz externa, como la estructura interna, del módulo, la cual se describirá en el próximo apartado.

A su vez el bloque es parametrizable mediante tres *generics*: Frecuencia del sistema clk_i (*FREC_CLK*), velocidad de comunicación promedio deseada (*BAUD*), y resolución en bits del contador fraccional (*RES_DIV_FRACCIONAL*).

Así, se simplifica la utilización del bloque, ya que internamente se calculan los valores límites de cuenta de los contadores.

B. Estructura interna

El esquema presentado en la Fig. 1, es conceptual, para comprender el funcionamiento del divisor. Es viable su implementación, pero posee una carga importante de lógica combinacional: el comparador y el sumador. Como se expresa en [6] grandes grupos de lógica combinacional generan una disminución en el rendimiento temporal en las FPGAs. Es por tanto, que en la estructura implementada se buscó reducir este aspecto, haciendo uso de un contador sincrónico ascendente, *Contador_Enable*, de módulo genérico, que posee una señal de habilitación (*E*) y un señal de fin de cuenta (*tc_o*). Este bloque ya formaba parte del conjunto de bloques utilizados y extensamente probados por el grupo de trabajo. La implementación basada en éste bloque se presenta en la Fig. 3. La composición de la estructura posee un comparador, un registro de valor constante, un multiplexor de 2 a 1 y tres contadores, basados en el bloque *Contador_Enable*:

- *cont_fracc*: Divisor fraccional, cuenta de 0 a 2^M*-1*
- *cont_ppal*: Divisor entero, genera la salida de sobremuestreo, *clkbaudx16_o*
- *cont_div_16*: Divisor fijo por un valor de 16 veces, genera la salida *clkbaud _o*.

La estructura planteada elimina el sumador valiéndose de la salida de fin de cuenta del contador principal, *cont_ppal*. El concepto es que en vez de modificar en un pulso la cuenta del

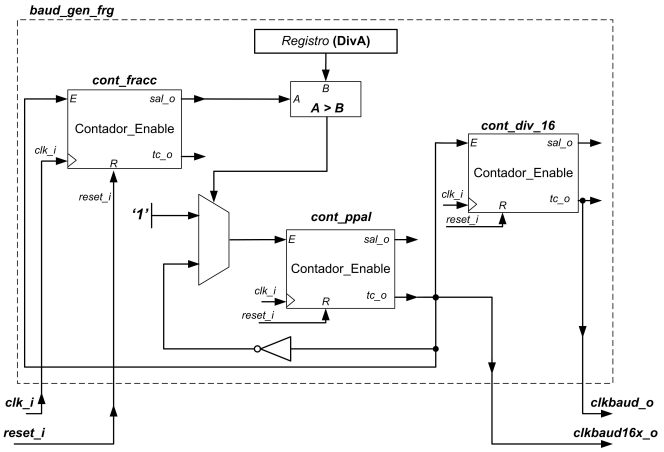


Fig. 3. Estructura interna del generador de *baud rate* .

contador principal se aprovechó que la señal de fin de cuenta pulsa durante un ciclo de reloj (*clk_i*) para indicar el final de la cuenta, realimentando esta señal al puerto de habilitación del contador principal a través del multiplexor comandado por el resultado de la comparación (según sea el valor de cuenta del contador fraccional *cont_fracc*), se inhibe (o no) durante un ciclo de reloj la cuenta principal. En caso de inhibirse, equivale temporalmente a contar un ciclo más de reloj. De esta manera la lógica combinacional se reduce de un sumador y un multiplexor de ancho *DivEnt+1* bits, a solamente un multiplexor de 1 bit y un negador.

C. Materiales y herramientas

Para realizar la implementación se siguió el flujo de trabajo habitual que se utiliza en el Laboratorio que consiste en: describir el bloque, evaluar los resultados del sintetizador, realizar las simulaciones de comportamiento, *testbench*, e implementar en *hardware* el bloque. Como lenguaje de descripción de *hardware* y para realizar la simulación se ha utilizado VHDL. Las simulaciones son de comportamiento en bancos de prueba automático, que verifica el valor medio y la desviación de las dos salidas. Se utilizó el IDE *ISE WebPACK®* 14.7. La plataforma utilizada para evaluar físicamente el bloque es un *kit* comercial [7], que contiene una FPGA de Xilinx Spartan 6, modelo XC6SLX25 [8], con un reloj principal de 50MHz y la circuitería mínima necesaria para el funcionamiento de la FPGA.

IV. Resultados

A. Síntesis

Los resultados de la síntesis en términos de recursos de *hardware*, se resumen en la Tabla. III, para distintas velocidades de comunicación. La máxima frecuencia de operación resultante en todos los casos es superior a los 50MHz de frecuencia del sistema. Se observa que conforme aumenta la velocidad de comunicación disminuye el uso de recursos y aumenta la frecuencia máxima admisible. Esto es lógico ya que el incremento en la velocidad de comunicación implica que

TABLE III
RESULTADOS DE LA SÍNTESIS

Baud rate bps	Slice LUTs	Slice Registers	BUFGP	fmax MHz
230400	14	10	10	366
460800	11	9	9	366
921600	10	8	8	397

la relación entre la frecuencias de entrada-salida disminuya, y por tanto se reduce el tamaño del contador principal y la lógica combinacional asociada a éste, resultando un aumento del rendimiento temporal del conjunto.

B. Mediciones

Para verificar el funcionamiento del bloque en *hardware* se realizaron una serie de mediciones con el contador de frecuencia Tektronix ® FCA3000, que posee un módulo de análisis estadístico lo que para este caso es de suma utilidad. Los resultados de las mediciones se presentan en la Tabla. IV, siendo $fout_m$ el valor medio y σ el desvío estándar. En la última columna se calculó el error relativo, *Er*, para cada medición corroborando que es muy inferior al 2% necesario, y que se corresponde con los resultados teóricos calculados anteriormente (Tabla. II).

C. Verificación

Para realizar una verificación de funcionamiento sobre el *kit* de evaluación, se procedió a reemplazar en un bloque-*IP* UART propietario el divisor de tipo entero, por el generador *baud_gen_frg* basado en el divisor fraccional. Se realizaron las evaluaciones en simulación funcional y luego se evalúo sobre el *kit*. Se tomó como caso de prueba, una necesidad puntual del Proyecto el establecimiento de un vínculo entre dos computadoras, una de éstas oficiaba de transmisor de datos y la otra de receptor siendo la FPGA el enrutador. Internamente en la FPGA se instanciaron dos UARTs, ambas con divisor fraccional y otro bloque que acumulaba los datos recibidos y conforme ingresaban los despachaba hacia el otro extremo de la comunicación. En función de las necesidades propias de utilización para el Proyecto, los datos se enviaban en tramas fijas de 64 bytes, bajo dos condiciones: período de trama de 5 ms para un *baud rate* de 230400 bps, y con período de trama de 2 ms para una *baud rate* de 460800 bps.

El envío de datos se realizó mediante una aplicación (*software*) autónomo, que conforma una trama de datos compuesta de:un encabezado, un contador de tramas de 16 bits y *checksum* de 8 bits. El receptor verifica estos parámetros mediante otra aplicación, posibilitando determinar la pérdida y/o corrupción de tramas. En un contexto de laboratorio, no deberían existir pérdidas excepto que las partes tengan fallas de comportamiento. Las dos aplicaciones utilizadas son confiables, se han evaluado y verificado extensamente, por ende las únicas fallas podían radicar en las UARTs implementadas. Los resultados obtenidos de este proceso arrojaron 0% de pérdida o corrupción de las tramas, quedando verificado el bloque *IP* para las condiciones establecidas.

TABLE IV
RESULTADOS DE LAS MEDICIONES

fout Hz	$fout_m$ Hz	σ mHz	Er %
230400	230414,7109	1,1435	-0,0064
460800	458715,5536	1,0355	0,4524

V. CONCLUSIONES

Se diseñó, desarrolló y verificó un generador de *baud rate* basado en un divisor fraccional totalmente descripto en VHDL e implementado sobre una FPGA. Se comprobó que este tipo de divisor es una solución apropiada para el caso en el cual la frecuencia del sistema no es un múltiplo exacto del *baud rate* deseado y sobre todo en los casos en donde la diferencia entre las frecuencias de entrada - salida es inferior a 10 veces, siendo en ésta condición más apreciable el error por redondeo de un divisor entero. Se logró, así, una solución adecuada para una comunicación serie asíncrona con un costo relativamente reducido de recursos de *hardware* adicionales: un contador de 4 bits y una lógica combinacional mínima. Algo sumamente importante es que esta implementación brindó una solución para un Proyecto específico y no sólo se satisficieron las necesidades funcionales necesarias, sino que también se verificó el funcionamiento del divisor mediante mediciones de laboratorio con el instrumental apropiado. El módulo generador desarrollado presenta un grado interesante de abstracción para el usuario al calcular internamente el valor límite de cada contador, en función de sólo dos parámetros: la velocidad de comunicación deseada y la frecuencia principal del sistema. Como trabajo a futuro se ha detectado que sea factible modificar durante la operación el valor de la velocidad de comunicación, esto ampliaría el uso del generador.

ACKNOWLEDGMENT

Este trabajo fue realizado en la Laboratorio Técnicas Digitales de CITEDEF, en el marco del Proyecto PIDDEF 01/ESP/15 BAA. Agradeciendo la labor diaria a los miembros del laboratorio: Daniel Pastafiglia, Martín Morales, Sergio Saluzzi, Adrián Stacul, Sebastián Alvarez y Gerardo García.

REFERENCES

[1] S. Dhage, M.Patil, N. Temgire, P. Vaity and S. Parshionikar, "Design and FPGA Implementation of a High Speed UART," International Journal of Scientific and Engineering Research, Volume7 , Issue 9, September 2016.
[2] Philips Semiconductors, "AN10333. SC16CXXXB baud rate deviation tolerance," 2004.
[3] PC16550D *datasheet*. Texas Instruments. 2015. Available: http://www.ti.com/lit/ds/symlink/pc16550d.pdf
[4] N. Gupta and R. Goyal, "Programmable Fractional Clock Frecuency Divider Circuit," Freescale Semiconductor, unpublished.
[5] ST Semiconductors, "RM0383- Universal synchronous asynchronous receiver transmitter (USART)," 2014.
[6] Pong P. Chu, "RTL Hardware Design using VHDL," John Wiley and Sons, Inc, 2006, 150-154pp.
[7] Emtech, "Placa 3PX1, Manual de usuario," 2015.
[8] Xilinx, Spartan-6 FPGA Documentation. Available: http://www.xilinx.com/support/documentation/spartan-6.htm.

Implementación de un Osciloscopio en una FPGA

‡Fernando Poy, §Alejandro Radosta, *Facundo Aguilera y †Guillermo Magallan
Grupo de Electrónica Aplicada
Facultad de Ingeniería
Universidad Nacional de Río Cuarto
Río Cuarto, Argentina
‡ferpoy23@gmail.com, §aleradosta1996@gmail.com, *afacu@ieee.org, †g.a.magallan@ieee.org,

***Resumen*—El presente artículo consiste en el diseño e implementación de un osciloscopio digital en una placa de desarrollo con FPGA. El desarrollo presenta una plataforma de bajo costo, flexible, para la visualización de forma de onda de tensión. La implementación se llevó a cabo experimentalmente en un kit de desarrollo starter kit, del cual se utilizaron los integrados de amplificación de señal y conversión analógica/digital para la obtención de las muestras de señal que luego se almacenan en el FPGA, donde se realiza el próximo tratamiento digital para mostrar la forma de onda en un monitor de computadora a través de un puerto VGA.**

***Index Terms*—Osciloscopio, FPGA, VDHL, Implementación, Kit Spartan-3E.**

I. Introducción

La experimentación, diseño, implementación y pruebas de laboratorio de sistemas eléctricos y electrónicos, requiere de la utilización de diferentes instrumentos que aseguren la calidad de las actividades realizadas. Uno de los instrumentos más usados es el osciloscopio, elemento indispensable para la medición de formas de ondas temporales tanto eléctricas como de otra naturaleza.

Con el objetivo de implementar plataformas de desarrollo con FPGA para fines didácticos, se ha propuesto el diseño de osciloscopios digitales basados en FPGA [1]. Este tipo de desarrollos presenta la ventaja, frente a los osciloscopios comerciales, de ser una alternativa económica al aprovechar los kits de desarrollo en FPGA disponibles en los laboratorios. Además, al ser diseños propios, presentan la flexibilidad de permitir la incorporación de etapas de procesamiento específicas que no estén disponibles en osciloscopios comerciales, tales como estrategias de disparo avanzadas, análisis de variables eléctricas para diagnóstico de equipos o lectura de sensores con protocolos específicos [2]–[4].

Algunos diseños de osciloscopios propuestos están pensados como instrumentos virtuales, requiriendo el uso de computadoras y un software específico para la visualización de los resultados [5], [6]. Otras propuestas implementan el sistema completo en el FPGA, incluyendo el procesamiento de las señales y la generación de señales para visualización en un monitor [7], [8].

El presente trabajo surge a partir de una propuesta de trabajo final en la asignatura de grado ”Programación Lógica para Ingeniería”, en la carrera Ingeniería Eléctrica de la Universidad Nacional de Río Cuarto, se realizó el diseño, simulación y programación de un Osciloscopio Digital en lenguaje VHDL, sobre FPGA, haciendo uso de la plataforma Xilinx ISE. El kit que se utilizó para el desarrollo del dispositivo fué el SPARTAN 3E.

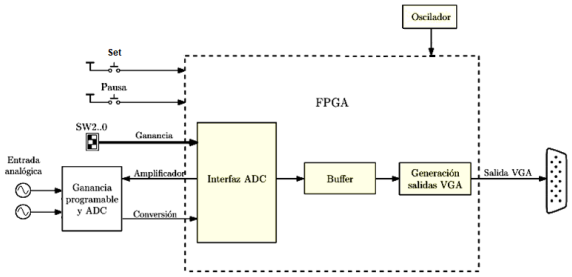


Figura 1. Diagrama simplificado.

El diseño se basó en un sistema de bloques sincronizados en el cual se adquieren datos de forma analógica, se amplifican y se convierten en digital a través de dos integrados disponibles en el kit y se procesan dentro del FPGA. Los datos son procesados en una memoria de doble puerto y luego se genera una señal para visualización a través de un puerto VGA. El osciloscopio implementa las funciones de disparo (trigger) y de pausa.

Es importante destacar los fines didácticos que se pueden atribuir a este proyecto, permitiendo un entendimiento más profundo del funcionamiento interno de un osciloscopio y resulta un diseño interesante para obtener una experiencia práctica de lo que involucra un proyecto de desarrollo digital basado en FPGA.

Este trabajo aprovecha los recursos de adquisición y puestos disponibles en el FPGA para reducir al mínimo la necesidad de componentes adicionales. Si bien el diseño realizado solamente incorpora las funciones básicas de un osciloscopio, la plataforma realizada permite la incorporación de algoritmos de procesamiento más avanzados en el futuro.

II. Diseño

El objetivo de este proyecto es el diseño de un osciloscopio digital, con características básicas, que sea flexible para permitir su utilización en aplicaciones específicas, como por

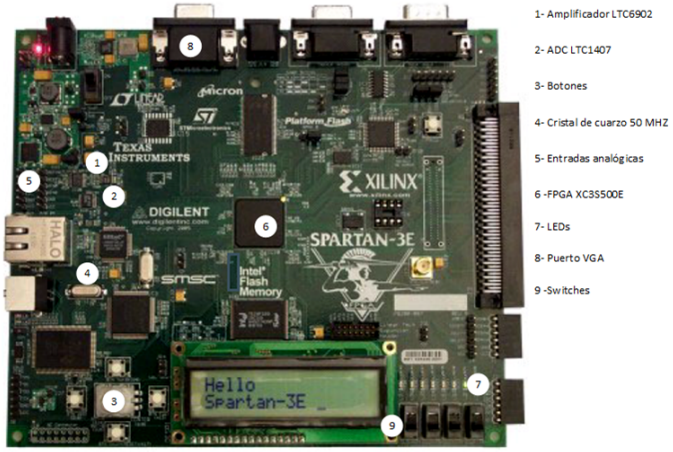


Figura 2. Spartan 3E Starter Kit.

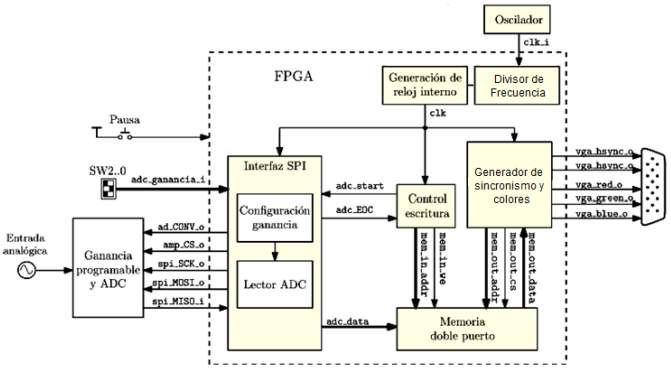


Figura 3. Diagrama de bloques del diseño en FPGA.

ejemplo mostrar la FFT o el valor RMS de una onda. En la Fig. 1 se muestra el esquema básico.

El diseño se basó en el aprovechamiento de los recursos disponibles en un kit Spartan-3E Starter Board [9]. Este kit posee un conjunto de periféricos para diferentes aplicaciones, lo que la convierte en una alternativa muy versátil. Para el diseño se hizo uso de los siguientes componentes que contiene el kit de desarrollo: amplificador de ganancia LTC6902, cristal de cuarzo de 50 MHz, conversor ADC LTC1407, y el FPGA de Xilinx XC3S500E que se vé en la Fig. 2. Los periféricos utilizados fueron: 4 switches, 4 botones, 8 led's, 4 pines de entrada y puerto VGA.

A través de los pines de entrada de señales analógicas se adquieren los valores de tensión a niveles admisibles por el amplificador de ganancia. El valor de ganancia de este amplificador es configurable digitalmente, mediante 3 bits, obteniendo 8 valores de selección de ganancia y seleccionándose con un botón pulsador (set), dando la orden de cambio. Luego se pasa a la etapa de conversión en la cual, mediante el chip ADC, se codifica la señal analógica a digital, obteniendo una palabra de 14 bits por canal.

Como se observa en la Fig. 3 el FPGA se comunica mediante el protocolo serie sincrónico SPI, tanto con el amplificador de ganancia programable como con el ADC, enviando el código de ganancia a los integrados y recibiendo el valor de cada muestra ya codificada. Dicha muestra se almacena en una memoria de doble puerto (disponible en el FPGA).

El diseño posee un bloque ïnterfaz gráfica"donde se genera una sincronización de la lectura de datos desde la memoria, y la muestra en el monitor a través de un puerto VGA.

El sistema diseñado posee dos canales y se selecciona entre uno y otro mediante una llave.

A continuación, se describen con mayor detalle las etapas de funcionamiento del osciloscopio.

II-A. Ganancia Configurable y Lector ADC

La primera etapa consiste en generar un bloque que se encarga de la comunicación entre el FPGA y la unidad de conversión analógica a digital que posee el kit de desarrollo Digilent Spartan-3E Starter. Esta unidad está formada básicamente de dos integrados, un amplificador de ganancia programable (PGA) LTC6902 y un ADC LTC1407, que poseen entradas para la adquisición de dos canales diferentes.

La comunicación entre el PGA y el FPGA se realiza a través de un bus SPI, enviando en tren de bits, con 2 códigos de 4 bits cada uno, con el nivel que se amplifica cada canal como se observa en la Tabla I. La señal amplificada ingresa al ADC que muestrea ambos canales, codificándolo en una palabra de 14 bits.

Esta sección se compone por dos bloques principales: *Configuración de ganancia* y *Lector ADC*. El primero lee los valores de entrada para cada canal a través de los buses internos de 4 bits `ganancia_A_i` y `ganancia_B_i`, siempre que se active la señal `ganancia_set_i`.

Luego, desde el PGA se envían los datos de las dos configuraciones de ganancia, correspondientes a cada uno de los canales, usando el bus SPI, a través de las líneas `amp_CS_o`, `spi_SCK_o` y `spi_MOSI_o`. Por la señal `spi_MISO_o` se recibe del integrado los dos códigos de ganancia establecidos, verificando la configuración correcta. En etapas de diseño, se usó un switch del kit de desarrollo para mostrar los códigos de ganancia configurados a través de LEDs, con el fin de verificar la comunicación.

El bloque *Lector ADC* recibe los datos enviados por el ADC que dispone la placa a través del bus SPI, usando las líneas `ad_CONV_o`, `spi_SCK_o` y `spi_MISO_i`. Los datos recibidos de cada canal son colocados en los buses internos de 14 bits `adc_chA_o` y `adc_chB_o`. Cada vez que se adquiere un dato completo desde el ADC, se activa durante un ciclo de reloj la señal `adc_ready_o`. El bloque *Lector ADC* se desactiva cuando se está utilizando el bloque de Configuración de ganancia. En la Fig. 4 se observa el diagrama de bloques de ganancia y ADC.

En etapas de diseño, para asegurar que el dispositivo realiza una correcta adquisición y conversión, se envían los 8 bits más significativos de una de las muestras al conjunto de ocho LEDs, permitiendo verificar si la codificación digital es la deseada.

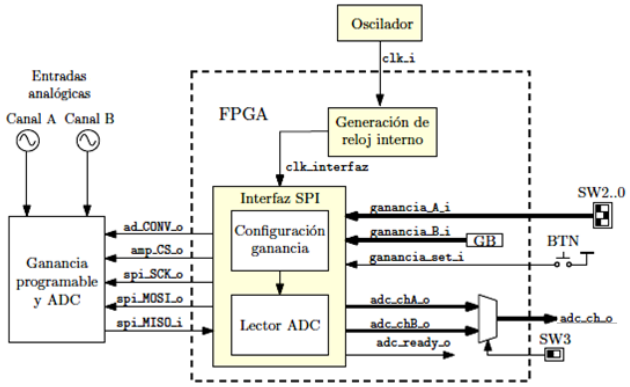


Figura 4. Diagrama de bloques Ganancia y ADC.

Tabla I
AJUSTES DE GANANCIA PROGRAMABLE DEL AMPLIFICADOR

GANANCIA	A3	A2	A1	A0	RANGO DE TENSION	
	B3	B2	B1	B0	Mínimo	Máximo
0	0	0	0	0	-	-
-1	0	0	0	1	0.4	2.9
-2	0	0	1	0	1.025	2.275
-5	0	0	1	1	1.4	1.9
-10	0	1	0	0	1.525	1.775
-20	1	0	0	1	1.5875	1.7125
-50	0	1	1	0	1.625	1.675
-100	0	1	1	1	1.6375	1.6625

II-B. Almacenamiento de datos

La segunda etapa consiste en almacenar los datos que son codificados por el ADC. Se decidió utilizar una memoria RAM de doble puerto con capacidad de lectura y escritura a frecuencias de reloj diferentes, para poder regular la velocidad con la que se realizan las tareas de la tarea de almacenamiento de datos muestreados y de visualización en pantalla de forma independiente.

La memoria de doble puerto fue conformada con BRAM disponible en el FPGA, motivo por el cual no se necesitó diseñar una memoria nueva optimizando la cantidad de espacio utilizado. Se seleccionó la memoria de un tamaño de la palabra de 14 bits de escritura (tamaño de cada muestra) y una profundidad de escritura de 24320 muestras, ocupando la disponibilidad total de bloques BRAM disponibles en el FPGA utilizada. En la etapa de lectura de la memoria, se mostrará un conjunto de 640 muestras, obteniendo un total de 38 conjuntos de 640 muestras como se verá en la Sección II-C. De esta manera, intercalando la posición en la que se leen las muestras de la memoria, se logra ajustar el canal horizontal o de tiempo.

II-C. Señales de sincronismo y colores

Como etapa final, el bloque *Generador de sincronismo y colores* se encarga de la etapa correspondiente a la generación de las señales de sincronización del puerto VGA y las señales de colores que dibujan en pantalla la forma de onda, grilla y lecturas con el nombre del canal que se está mostrando. En la Fig. 3 se puede observar este bloque.

Para la visualización se definió una resolución soportada por la mayoría de los monitores de 640x480 pixeles. En el eje vertical se representó la altura de la onda muestreada, asignando a cada pixel un grupo de muestras obtenido del cociente entre la cantidad de valores distintos que puede adquirir el ADC con una palabra de 14 bits (2^{14}) y los 480 niveles de altura. El canal horizontal se tomó como el eje temporal, visualizando 640 muestras.

La comunicación con el monitor se realizó con un puerto VGA, por el que se envían 5 señales:

- `vga_hsync_o` y `vga_vsync_o`, encargadas de la sincronización de los barridos horizontal y vertical de la pantalla correspondiente a un reseteo de imagen a 60 Hz.
- `vga_blue_o`, `vga_green_o` y `vga_red_o`, generan la combinación de colores necesaria para crear la imagen en pantalla.

Como la muestra de una imagen en pantalla se hace a través de barridos horizontales, de forma sincrónica con los canales `vga_hsync_o` y `vga_vsync_o` se toma cada pixel vertical como un nivel, de tal modo que por cada barrido horizontal se leen las 640 muestras y se las compara con dicho nivel, pintando de color verde para el canal 1 y rojo para el canal 2 (dependiendo de cual esté seleccionado para mostrar) cuando la muestra pertenece a ese nivel y de color blanco (color de base) cuando no corresponde. Los valores de tiempos entre cada barrido, se obtuvieron de datasheet del kit utilizado.

Además se agregó una grilla como la que se observa en la Fig. 6 para las mediciones de tensión de dirección vertical y horizontal de forma gráfica. Por pantalla también se visualizan las configuraciones seleccionadas desde la interfaz de usuario cuando se pulsa algún botón de la placa. Y por último para identificar el canal en pantalla se colocó el nombre del canal también en la parte inferior izquierda.

II-D. Divisor de Frecuencia

Uno de los bloques necesarios para la implementación del Osciloscopio es el *Divisor de Frecuencia*. Este bloque, a partir de la señal de reloj de 50 MHz que posee el kit de desarrollo, establece las frecuencias de operación de los componentes externos (ADC y PGA) y los bloques internos del FPGA, mediante la utilización de contadores y un DCM disponible dentro del FPGA. Con esto, se establecen las frecuencias de las señales de reloj (Clock), según las distintas necesidades.

El bloque *Ganancia Configurable* y *Convertidor ADC* que se detalló en la Sección II-A se diseñó con una frecuencia de Clock de 25 MHz, máxima admisible por los integrados utilizados para esta tarea.

En el bloque Almacenamiento que se explicó en la Sección II-B, los datos se escribieron a 1.25 MHz y en la etapa de visualización por pantalla mencionada en la Sección II-C dichos datos se leyeron a 25 MHz.

Cabe mencionar que todos los bloques internos comparten la misma señal de reloj, mientras que las diferentes frecuencias de operación requeridas se establecieron a partir del uso de contadores que generan señales de habilitación.

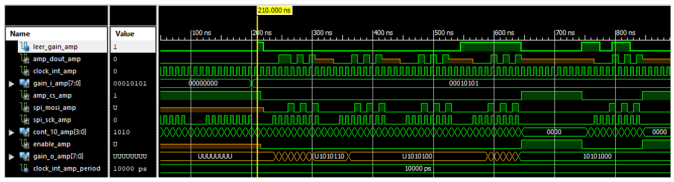


Figura 5. Test Bench - Bloque Amplificador de Ganancia.

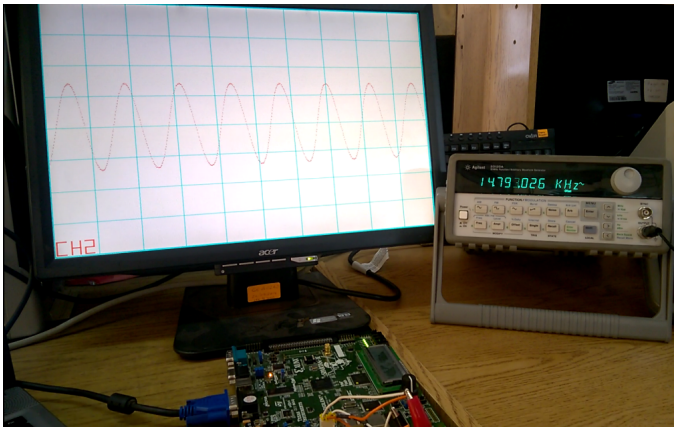


Figura 6. Verificación de resultados.

Cada señal de reloj se eligieron adecuadamente de manera que las tareas de cada bloque se ejecuten de forma independiente, pero sin perder sincronismo entre ellos.

III. Verificación

Una vez que se realizó el diseño en FPGA, se realizaron simulaciones mediante una serie de Test Bench para verificar el funcionamiento de las diferentes etapas. Una de las simulaciones más importantes realizadas fue la configuración de ganancia, donde se necesitó que la carga del nivel de amplificación de la señal de entrada sea la correcta. Como se observa en Fig 5, se simuló la señal recibida por el botón pulsador de activación de ganancia, lográndose ver como se envían correctamente los datos por las señales correspondiente hasta identificar que se deshabilita el integrado al lograr una configuración exitosa.

Cuando los resultados de simulación fueron satisfactorios, con la utilización de un generador de señales que se muestra en la Fig. 6 se procedió a realizar ensayos de muestreo de datos y visualización en pantalla de ondas conocidas con características predeterminadas para lograr de manera sencilla identificar errores o inconvenientes para luego corregirlos.

Usando un Osciloscopio comercial y un generador de funciones, se generó una onda de tensión adecuada para cada valor de ganancia programable, y se lo comparó con el osciloscopio diseñado en FPGA, logrando corroborar configuraciones correctas y los mismos valores característicos de las señales, verificando un correcto funcionamiento. Los distintos valores de ganancia se observan en la Tabla I.

IV. Conclusiones

El dispositivo diseñado se consideró de gran utilidad como un osciloscopio comercial, con la ventaja de poder disponer del acceso de datos para un posterior adaptación a ciertas aplicaciones específicas si se lo desea.

Si en un futuro se necesita incorporar una aplicación específica como una operación matemática por ejemplo una FFT (Fast Fourier Transform), este tipo de diseño presentaría la ventaja de ser adaptable, con la capacidad de observar en pantalla la obtención de algún valor específico buscado.

Bajo fines didácticos se logró adquirir conocimientos nuevos y necesarios para ser utilizados en otras asignaturas y en nuestro futuro profesional.

Referencias

[1] B. Başa and M. İskefiyeli, "Realization of Digital Oscilloscope with FPGA for Education," *Procedia - Social and Behavioral Sciences*, vol. 174, pp. 814–820, Feb. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187704281500717X

[2] S. A. Khan, A. K. Agarwala, D. T. Shahani, and M. M. Alam, "Advance Oscilloscope Triggering," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 3, pp. 944–953, Jun. 2007. [Online]. Available: http://ieeexplore.ieee.org/document/4201004/

[3] R. J. Costa, D. E. Pinho, and G. R. Alves, "Using embedded instruments to design weblabs: An FPGA-embedded oscilloscope based on the IEEE1451.0 Std." in *2015 3rd Experiment International Conference (exp.at'15)*. Ponta Delgada, Portugal: IEEE, Jun. 2015, pp. 41–46. [Online]. Available: http://ieeexplore.ieee.org/document/7463211/

[4] M. Otero, M. G. Scolari, P. M. d. l. Barrera, and G. R. Bossio, "Detección de fallas en barras de un motor de inducción utilizando inyección de señales de secuencia cero," in *2016 IEEE Biennial Congress of Argentina (ARGENCON)*, Jun. 2016, pp. 1–6.

[5] F. Aguilera, C. S. Paez, and D. Costa, "Implementación de un osciloscopio en una plataforma de instrumentación virtual reconfigurable," in *XXII Congreso Argentino de Control Automático*, 2010.

[6] P. Miranda Loza and F. G. Ávila Elías, "Diseño de un osciloscopio multicanal con FPGA (Proyecto Lago)," *Revista Boliviana de Física*, vol. 17, no. 17, pp. 27–31, 2010.

[7] F. Vicedo, M. Garcia, A. Grediaga, S. Alcaraz, P. Garrido, and K. G. Llamazares, "Low cost digital oscilloscope." *WSEAS Transactions on Circuits and Systems*, vol. 3, no. 9, pp. 2030–2034, 2004.

[8] N. Mehmood, J. Ogniewski, and V. Ravinath, "Real-time digital oscilloscope implementation in 90nm CMOS technology FPGA," *World Academy of Science, Engineering and Technology, International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, vol. 4, no. 11, pp. 1645–1648, 2010.

[9] Xilinx Inc., *Spartan-3E FPGA Starter Kit Board User Guide*, 1.1 ed., Jun. 2018.

Control para convertidor analógico-digital de aproximaciones sucesivas

María Isabel Schiavon, Daniel Crepaldo, Carlos Varela, Lisandro Martín.
Laboratorio de Microelectrónica - FCEIA - Universidad Nacional de Rosario
Rosario, Argentina
microlab@fceia.unr.edu.ar

***Abstract*— Se presenta la verificación de funcionalidad del control de un convertidor analógico-digital de aproximaciones sucesivas por reparto de carga mediante su implementación en lógica programable como paso previo a su diseño e implementación en tecnología CMOS. Este convertidor se utilizará para obtener en valores binarios la tensión de salida de sensores incluidos en los nodos de una red de recolección de datos inteligente reconfigurable que se utiliza para monitorear variables climáticas tales como temperatura, humedad, presión, etc. Los sensores entregan a la salida un valor de tensión proporcional a la magnitud medida, el cual debe ser muestreado, cuantificado y codificado para efectuar el almacenamiento y posterior transmisión de los datos a través de la red. Los nodos se están implementando como ASIC en tecnología CMOS.**

Keywords*— Convertidor A/D, mínimo consumo, tasa conversión, tecnología CMOS.

I. INTRODUCCIÓN

Se pretende realizar la conversión de la tensión de salida de los sensores incluidos en los nodos de una red inalámbrica inteligente reconfigurable encargada de monitorear y recolectar datos de variables climáticas (temperatura, humedad, presión, etc.) en campo [1] mediante un convertidor analógico-digital. Los sensores entregan a la salida un valor de tensión proporcional a la magnitud medida, el cual debe ser muestreado, cuantificado y codificado para efectuar el almacenamiento y posterior transmisión de los datos a través de la red. Todo el conjunto del nodo se va a implementar en CMOS como un sistema monochip.

La elección de la topología de conversión a utilizar se define a partir de un estudio comparativo entre las posibles opciones aplicando como criterios de selección las restricciones que impone la aplicación [2]:

- Mínimo consumo. A fin de maximizar la duración de las baterías, la reducción del consumo es una de las principales restricciones del diseño.
- Baja tasa de conversión. La baja variabilidad en el tiempo de las magnitudes a medir permite trabajar con una tasa de conversión por unidad de tiempo reducida.
- Resolución de conversión. Tomando como ejemplo típico la medición de temperatura, la precisión necesaria (0,1 ºC en el rango de temperatura ambiente) requiere un conversor con una resolución mínima de 10 bits.

Teniendo en cuenta estas restricciones se optó por realizar la conversión mediante aproximaciones sucesivas por reparto de carga. Esta opción puede trabajar sin problemas con la resolución planteada y a la tasa de conversión requerida por la aplicación, mientras que a partir de un estudio preliminar se determinó que su consumo energético es el mínimo dentro de las opciones disponibles.

A continuación, se detallan las características principales de esta topología.

Convertidor de aproximaciones sucesivas (SAR)

El convertidor de aproximaciones sucesivas, cuyo diagrama en bloques se presenta en la Fig. 1, realiza la conversión bit a bit comenzando desde el más significativo.

En un primer momento se fija el valor del MSB del registro de aproximaciones sucesivas (SAR) a uno, lo que lleva la salida del convertidor digital-analógico (DAC) a un valor de tensión ubicado en la mitad del rango de medición. Si la tensión de entrada es mayor, el bit se deja en 1, de lo contrario se cambia a 0. Este proceso se repite para el resto de los bits del SAR hasta obtener la salida deseada de n bits.

Existe una variante de esta topología pensada para reducir el consumo de energía en la cual las aproximaciones sucesivas se realizan modificando el balance de carga de un banco de capacitores [3], en la Fig. 2 se muestra el diagrama en bloques correspondiente. El banco de capacitores consta de n+1 capacitores siendo n el número de bits de resolución del convertidor. El primer capacitor tiene un valor C, el segundo un valor C/2, el tercero C/4 y así sucesivamente, hasta los dos últimos, cuyo valor será $C/2^{n-1}$.

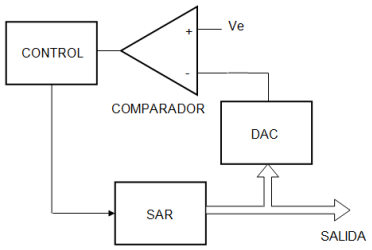


Fig. 1. Convertidor de aproximaciones sucesivas.

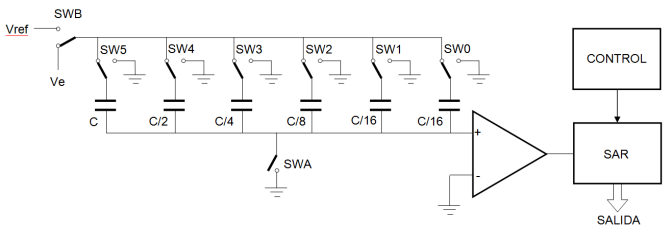


Fig. 2. Convertidor SAR de redistribución de carga.

978-1-7281-1363-0/19/$31.00 ©2019 IEEE

Un juego de llaves analógicas permite conectar el borne superior de estos capacitores a la tensión de entrada, a una tensión de referencia o a masa, y la entrada no inversora del operacional que coincide con el borne inferior de los capacitores a masa. En la fase inicial de la conversión todos los capacitores se cargan con Ve, luego el primer capacitor se conecta a Vref y el resto a masa, por lo que se constituye un circuito serie entre C y el resto de los capacitores conectados en paralelo. En la ecuación (1) se ve la expresión de la tensión a la entrada del comparador :

$$V_{entrada\ comparador} = -V_e + V_{ref}/2 \qquad (1)$$

Dependiendo de si este resultado es positivo o negativo se determina el valor del primer bit, y también se define la posición de la llave del primer capacitor para el siguiente paso. Si el primer bit fue 1 el capacitor 1 se conecta a masa, de otra manera se deja conectada a V_{ref}. El proceso continúa conectando en paralelo el siguiente capacitor (C/2) con el resto, con lo que la tensión a la entrada del comparador en el caso de que el primer bit haya tenido un valor alto se expresa en la ecuación (2).

$$V_{entrada\ comparador} = -V_e + (3/4)V_{ref} \qquad (2)$$

Así sucesivamente durante n ciclos hasta que se haya determinado el valor de todos los bits.

En el presente trabajo se presenta el diseño e implementación en lógica programable del control del convertidor analógico-digital, a fin de verificar el correcto funcionamiento de la topología seleccionada como paso previo a su implementación en tecnología CMOS.

II. DESARROLLO

En la Fig. 3 se muestra el diagrama de flujo correspondiente al funcionamiento del control del convertidor. Las llaves que controlan la carga de los capacitores (SW0 a SWn, SWA y SWB) se encuentran inicialmente conectadas a masa Para realizar una conversión, una vez recibida la señal de inicio (CONVERTIR) se pasa al estado de inicialización en el cual se conecta la llave SWB a Ve y se conmuta el estado de las llaves SW0 a SWn por lo que todos los capacitores se cargarán a la tensión de entrada. Estas llaves conmutadoras se implementan mediante dos puertas de transmisión conectadas con un terminal en común, las cuales se activan por separado de manera tal de evitar el solapamiento de los estados de conducción. Esto implica que se deben controlar 2(n+1) llaves, más dos llaves para el conmutador SWB y una para el interruptor SWA. Por lo tanto, para una resolución de n=10 bits la señal de control de las llaves del circuito (LLAVES) tendrá una longitud de 25 bits. Los bits menos significativos corresponden a las llaves SW0 a SWn, luego un bit de control de SWA y dos para SWB.

Concluida la carga inicial de los capacitores se continúa con la conversión abriendo SWA, conectando el capacitor de mayor valor a Vref y el resto a masa. Si $V_{entrada\ comparador} > 0$ (señal COMPARADOR = 1) significa que $V_i < V_{ref}/2$, por lo que el bit más significativo del registro de salida (SARn) debe ser cero y en el siguiente paso el capacitor de mayor valor debe permanecer conectado a masa. Caso contrario el bit correspondiente se fija en uno y el capacitor correspondiente se conecta a masa. El proceso continúa hasta completar todos los bits del registro de salida. En este momento se activa una señal para indicar que el resultado de la conversión está disponible (DATO_DISPONIBLE) y se pasa al estado de espera de la próxima activación de la señal CONVERTIR.

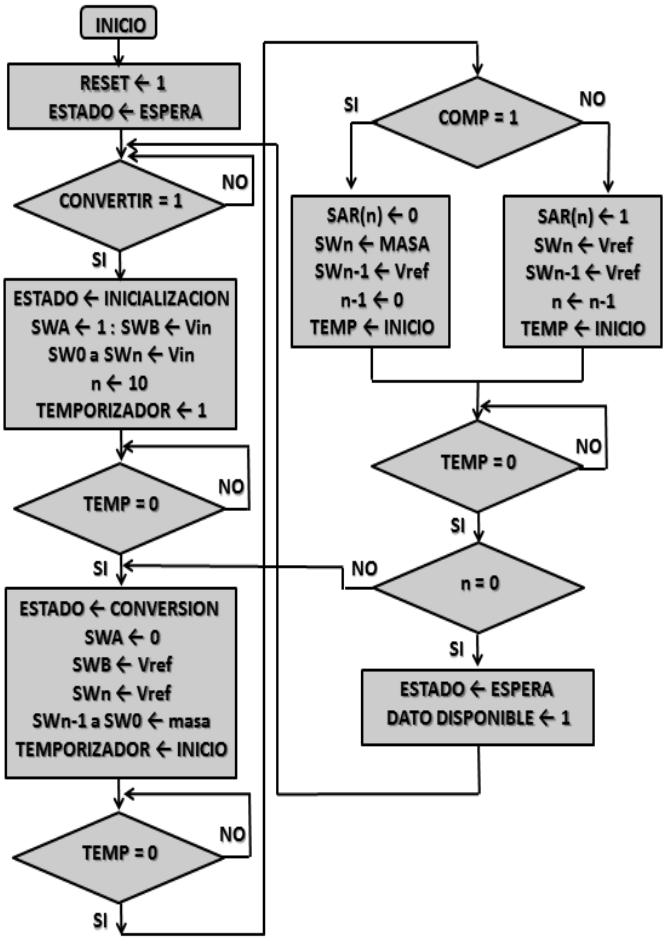


Fig. 3. Diagrama de flujo.

En la Fig. 4 se muestra un diagrama en bloques del sistema. El mismo está compuesto por tres bloques, uno encargado del control del proceso de conversión (CONTROL), uno cuya función es accionar las llaves analógicas que controlan la carga de los capacitores (CONTROL_LLAVES) y finalmente el registro de aproximaciones sucesivas (SAR).

El bloque de control se implementó como una máquina de estados finitos con tres estados: espera, inicialización y conversión. En estos dos últimos estados se generan las distintas configuraciones a asumir por las llaves de los capacitores a medida que se avanza en el proceso de conversión. Estas configuraciones se envían al bloque CONTROL_LLAVES a través de la señal DATOS de 25 bits. Este bloque es el encargado de evitar el solapamiento de los instantes de conexión y provee las temporizaciones necesarias para el funcionamiento de las llaves conmutadoras. Dado que la carga de los capacitores no es instantánea, es fundamental proveer los retardos de tiempo necesarios para permitir la estabilización del circuito en cada modificación de la posición de las llaves, lo que se logra efectuando una cuenta regresiva mediante un contador en cada ocasión en que se reciba un pedido a través de la señal SET. Modificando el valor de inicio de la cuenta se puede ajustar el tiempo de retardo al valor más adecuado. La señal DISPONIBLE se mantiene en bajo mientras dura esta temporización evitando que ingresen nuevos pedidos de configuración.

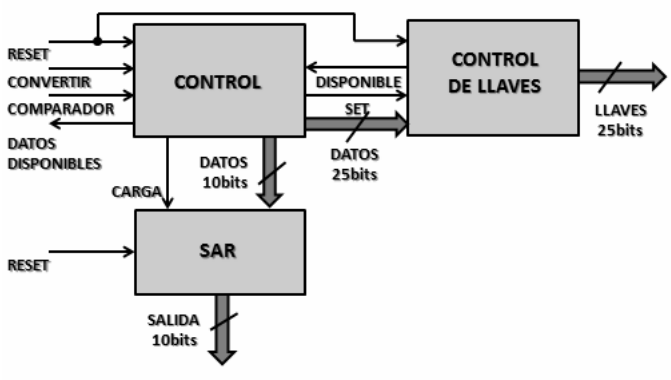


Fig. 4. Diagrama de bloques.

A medida que se progresa en la conversión se va actualizando el valor de los bits del registro SAR de acuerdo a los datos que va recibiendo del bloque de control a través de la señal DATO. Al terminar la conversión se activa la señal DATO_DISPONIBLE para indicar la presencia de un valor válido en el registro.

El prototipo del circuito de control se implementó mediante una FPGA SPARTAN3 montada en una placa de desarrollo Digilent Nexys 2 [4]. Se utilizaron 78 slices de un total de 4656 disponibles.

III. Resultados de Simulación

En la Fig. 5 se observan los resultados de la simulación del circuito. Puede apreciarse la conmutación de las llaves a partir de la llegada de la señal de inicio, durante el período de inicialización y el comienzo de la conversión.

En los períodos que se observan entre distintas configuraciones de llaves se las mantiene a todas abiertas para evitar solapamientos. Puede verse también cómo se van incorporando los bits del registro de salida.

IV. Conclusiones

Se diseñó e implementó utilizando lógica programable un circuito capaz controlar el funcionamiento de un convertidor analógico-digital actuando sobre las llaves que controlan la carga de un banco capacitores. La implementación se realizó sobre una placa de desarrollo Digilent Nexys 2 que contiene un dispositivo Spartan3 de Xilinx. El sistema presenta un funcionamiento acorde con los requerimientos planteados.

En una instancia posterior se planea construir el circuito completo del convertidor para evaluar su funcionamiento y ajustar las temporizaciones requeridas.

V. Referencias

[1] M. I. Schiavon, D. Crepaldo, "Autonomous wireless intelligent network accessible via IP" 7th Southern Conference on Programmable Logic (SPL´11), Córdoba, Argentina. Abril 2011.

[2] M. I. Schiavon, D. Crepaldo, C. Varela. "Análisis comparativo de topologías de convertidores analógico-digitales". IX Congreso de Microelectrónica Aplicada UEA2018. Catamarca, Argentina. Octubre 2018.

[3] T. Kugelstadt, "The operation of the SAR-ADC based on charge redistribution". Analog Applications Journal, Texas Instruments Incorporated, febrero 2000

[4] https://reference.digilentinc.com/reference/programmable-logic/nexys-2/reference-manual

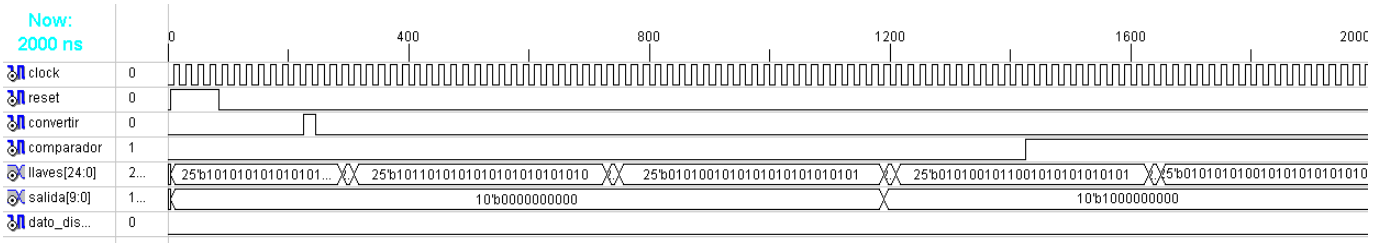


Fig. 5. Resultados de simulación

Kéfir, MILK y Lattuino: un Arduino reconfigurable

Salvador E. Tropea
Centro de Micro y Nano Tecnologías
Instituto Nacional de Tecnología Industrial
Buenos Aires, Argentina
email: salvador@inti.gob.ar

***Resumen*—En este trabajo presentamos un ecosistema completo compatible con Arduino basado en FPGA. El mismo es abierto y toda la información de diseño se encuentra disponible en internet.**

El ecosistema se compone de tres grandes componentes:

Kéfir, que es una placa de FPGA basada en la HX4K de la familia iCE40 de Lattice (7680 LUTs)

MILK, que es una placa de configuración multipropósito, con soporte para JTAG, SPI, RS-232, etc.

Lattuino, que es una implementación en VHDL y Verilog de un procesador compatible con AVR. El mismo incluye los periféricos necesarios para implementar un Arduino UNO.

Todo el ecosistema fue desarrollado utilizando herramientas de *software libre*. Adicionalmente puede ser sintetizado y programado utilizando *software libre*.

***Index Terms*—FPGA, Arduino, JTAG, SPI, RS-232, FT2232H, soft-core, AVR**

I. Introducción

El ecosistema presentado en este trabajo fue pensado para disponer de herramientas abiertas y económicas que permitieran gran flexibilidad a la hora de modelar y enseñar tecnologías de lógica programable. Al mismo tiempo se las pensó para que pudieran ofrecer un amplio espectro de posibilidades, de manera tal que su uso no se limite únicamente a la lógica programable.

Todas las placas pueden ser fabricadas con tecnologías simples. Los circuitos impresos son doble faz y los componentes elegidos pueden ser soldados manualmente.

Como el foco se puso en la lógica programable **Kéfir** [1] es la placa principal. Esta placa contiene una FPGA de bajo costo y la menor cantidad posible de componentes.

Para permitir un máximo reuso de los recursos se decidió separar la lógica de configuración de la FPGA. Es así que nace **MILK** [2]. Esta placa contiene la interfaz USB de configuración. La misma puede ser usada para otros propósitos ya que implementa SPI, JTAG y RS-232. De manera tal que esta placa puede usarse, por ejemplo, para depurar sistemas basados en procesadores ARM y AVR, o configurar FPGAs de Xilinx.

Con el objetivo de darle mayor versatilidad al conjunto se implementó una CPU compatible con el AVR utilizado en el Arduino UNO [3], junto con los periféricos y las bibliotecas necesarios. Estos componentes conforman lo que denominamos **Lattuino**. De esta manera es posible usar **Kéfir** y **MILK** con la simpleza de uso de Arduino, con la ventaja que podemos modelar nuestros propios periféricos y/o modificar la CPU, ya que son descripciones en HDL.

En la sección II describimos las características de la placa con FPGA, en la sección III la placa de configuración y en la sección IV la implementación compatible con Arduino UNO.

II. Kéfir, placa con FPGA

II-A. Selección de la FPGA

A la hora de seleccionar la FPGA se buscó que la misma fuera económica y que estuviera disponible en encapsuldos que se pudieran soldar manualmente. La familia iCE40 de Lattice cumple con estos requisitos, siendo la HX4K una FPGA de tamaño razonable y disponible en encapsulado *quad flat*.

Otras ventajas interesantes de esta FPGA son:

- Existen herramientas de *software libre* para desarrollar con la misma.
- No solo la FPGA es económica sino que su configuración se realiza desde una memoria SPI, cuyo costo es muy bajo.
- Si bien su fabricante la comercializa como una FPGA de 3520 LUTs, en realidad la pastilla que contiene es de 7680 LUTs. La limitación la realiza la herramienta del fabricante. Dicha limitación no existe al utilizar *software libre*.

II-B. Implementación

Desde su concepción se decidió que la placa debía ser lo más compatible posible con Arduino. Esto permitiría la posibilidad de utilizar las populares placas de expansión de Arduino (*shields*).

A los fines de maximizar la compatibilidad no solo se incluyeron las líneas digitales, sino que también se agregaron entradas analógicas. Se seleccionó el conversor MCP3008, por poseer características similares al del AVR.

Además de los conectores de expansión compatibles con Arduino se agregaron dos conectores tipo PMOD, permitiendo así el uso de periféricos de la empresa Digilent.

Para mantener el costo bajo se decidió no incluir demasiados LEDs y/o interruptores. Tomando como modelo la placa iCE Blink de Lattice se decidió incluir cuatro pulsadores capacitivos y cuatro LEDs.

Debido a que nuestro equipo de trabajo poseía experiencia con la implementación de USB en FPGAs [4] se decidió incluir un adaptador de niveles USB, el TUSB1106. El mismo es opcional, pudiendose puentear, lo cual funciona correctamente en condiciones de laboratorio.

Para el diseño se utilizó la herramienta de *software libre* KiCad [5]. En la Fig. 1 se observa la placa ya terminada.

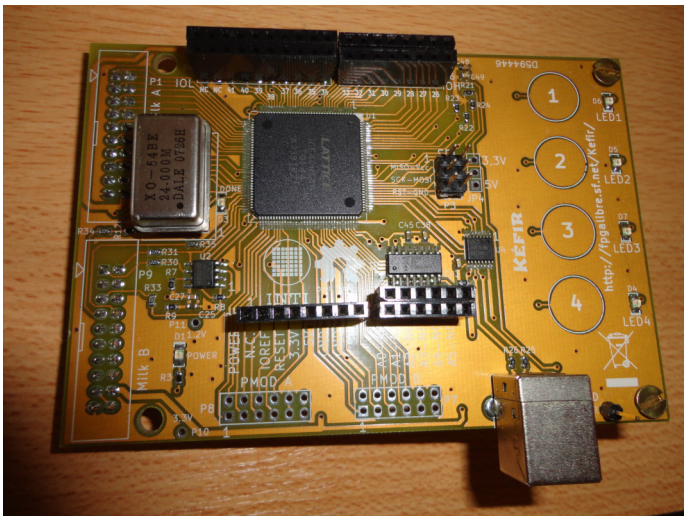


Figura 1. Placa Kéfir I

II-C. Características

- FPGA:
  - Lattice iCE40HX4K-TQ144 (7680 LUT4s, FFs y carry; 2 PLLs; 80 kib SRAM; 107 GPIOs)
  - Memoria de configuración: Winbond W25X40CL (SPI, 4 Mib, quedan casi 3 Mib libres)
  - Reloj: zócalo para colocar un oscilador tipo DIP-8/14. (evitando usar 1 PLL todo el tiempo)
- PCB:
  - Estándar: FR4 1,6 mm de espesor
  - Capas: 2
  - Separación mínima: 8 mils (fijada por el paso de la FPGA 0,5 mm)
  - Ancho mínimo de pistas: 10 mils
  - Diámetro vías: 40 mils
  - Agujero vías: 0,51 mm
- Conectores:
  - GPIO 3,3 V (la FPGA no tolera 5 V)
  - Tiene los conectores de ChipKit, que son los de Arduino UNO, pero con el doble de I/Os. Un total de 42 GPIOs.
  - 2 PMOD de 12 pines (8 I/Os + 2 GND + 2 Vcc c/u)
  - 2 conectores de 20 pines para configuración y comunicación con la PC. Estos conectores están pensados para conectar con **MILK**.
- Interfaz de usuario:
  - Botones: ninguno físico, tiene 4 CapSense.
  - LEDs: 4 LEDs de uso general (+ power + DONE).
- Entradas analógicas:
  - 8 canales de 10 bits
  - Frecuencia máxima de sampleo 100 ks/s
- USB:
  - Conector tipo B
  - Driver TUSB1106 (LS/FS)
  - Adicionalmente **MILK** provee comunicación con la PC
- Alimentación:
  - Desde el USB a través de **MILK**
  - Opcionalmente puede alimentarse con 3,3 V

III. **MILK**, CABLE DE CONFIGURACIÓN

La configuración de las iCE40 se realiza utilizando el protocolo SPI. Esto es tanto cuando se configura la memoria externa, como cuando se desea guardar la configuración en la memoria SRAM interna de la FPGA.

Las soluciones más comunes a este problema son utilizar un adaptador USB a SPI, o bien utilizar un microcontrolador con interfaz USB. La primer solución es la más simple, ya que el software de Lattice soporta adaptadores basados en el FT2232H [6]. La segunda solución es más económica, pero implica el uso de un software específico para transferir la configuración.

Es muy común que los kits de desarrollo incluyan el chip adaptador, por ejemplo el FT2232H. El problema con esta estrategia es que el costo del chip es tan alto como el de la FPGA. Para mantener el proceso de configuración lo más simple posible, pero sin desperdiciar recursos se decidió crear **MILK**.

MILK (Multiple Interface Light Kit) es una placa separada que contiene el FT2232H. De esta manera la configuración se puede hacer con las herramientas genéricas, tanto de Lattice como de software libre, y al mismo tiempo se puede aprovechar su funcionalidad para otras tareas de desarrollo.

El FT2232H soporta varios protocolos serie en forma nativa, pudiendo incluso implementar otros protocolos. Los modos soportados incluyen SPI, JTAG, RS-232 e I2C.

Este chip incluye dos serializadores, con lo que es posible implementar dos comunicaciones separadas. Cada serializador es un canal de comunicación independiente. Usualmente los kits utilizan un canal para configurar la FPGA y otro para comunicarse con algún *core* sintetizado en la FPGA.

Para darle mayor versatilidad **MILK** incluye adaptadores de niveles de tensión en uno de sus canales. De manera tal que es posible trabajar con tensiones entre 0,95 V y 5 V. Este es el único agregado, permitiendo utilizar los dos canales para cualquiera de los protocolos disponibles. Esto no es común en otros circuitos similares, que usualmente restringen el uso de cada canal a uno o algunos de los protocolos.

Para implementar los diferentes protocolos **MILK** se basa en un esquema de plug-ins. Así es posible implementar canales RS-232 con tensiones bipolares, basta con agregar el plug-in adecuado. Algunos plug-ins son simplemente cables con conectores, otros incluyen algún circuito electrónico.

En la Fig. 2 se observa la placa ya terminada.

III-A. Uso con Kéfir

Cuando **MILK** se usa con **Kéfir** provee:

- Configuración de la memoria SPI

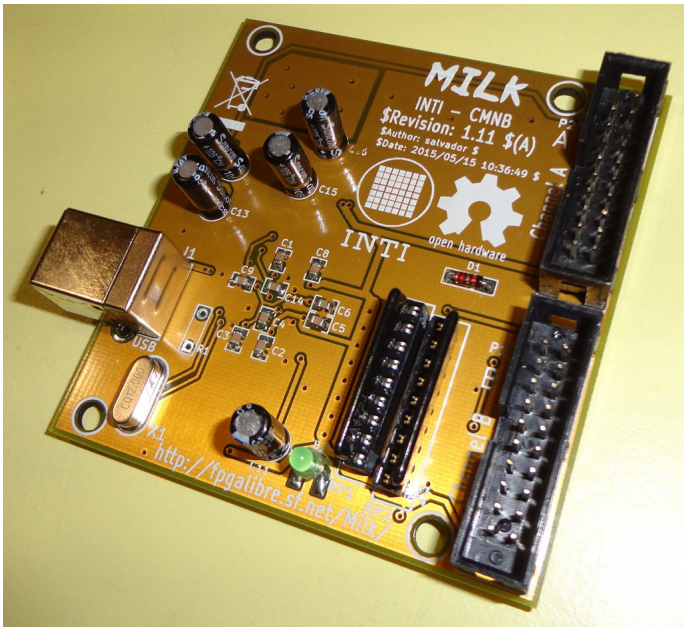


Figura 2. Placa MILK

- Comunicación con algún *core* en la FPGA. Pudiendo incluso implementar los modos de alta velocidad del FT2232H que permiten transmitir hasta 48 MB/s.
- Alimentación de 3,3 V, obtenida desde el puerto USB.

III-B. Otros usos

Se implementaron las siguientes aplicaciones:

- Comunicación genérica RS-232 con niveles bipolares y lógicos.
- Protocolo JTAG para
  - Uso genérico con UrJTAG [7]
  - Depuración de CPUs ARM con OpenOCD [8]
  - Configuración de FPGAs de Xilinx con iMPACT. Incluyendo modos de alta velocidad ya que **MILK** puede funcionar como un cable compatible con el Digilent HS1.
- Protocolo SPI para
  - Lectura y/o escritura de memorias SPI con flashrom [9]
  - Configuración y depuración de CPUs AVR con AVRDUDE [10]
  - Configuración de FPGAs de Lattice con Diamond Programmer

IV. Lattuino, un Arduino reconfigurable

Tal como se mencionó en la sección II-B **Kéfir** fue diseñada con la posibilidad de poder funcionar de manera similar a un Arduino UNO. Así es que nace Lattuino.

Lattuino es el conjunto de descripciones de hardware, bibliotecas de software y archivos de configuración necesarios para utilizar a **Kéfir** desde la interfaz de usuario del proyecto Arduino.

Para lograr este objetivo fue necesario realizar la implementación de un *soft-core*. En este punto se optó por implementar una CPU compatible con la línea AVR, con la cual ya teníamos experiencia [11]. De esta manera es posible utilizar las herramientas más comunes de Arduino, e incluso aprovechar rutinas escritas en lenguaje ensamblador.

La CPU es compatible con los ATmega, sin embargo algunas instrucciones necesitan de más ciclos de reloj. Esto es debido a que la arquitectura de los AVR implementa el banco de registros como una memoria asincrónica. Los bloques de memoria de las FPGAs son sincrónicos y esto obligó a introducir ciclos de espera. Otra opción hubiese sido no utilizar los bloques de memoria, pero en este caso las iCE40 necesitan demasiados recursos para su implementación.

Adicionalmente fue necesario implementar los periféricos de la CPU usada en el Arduino UNO. Se optó por implementar periféricos más simples que los originales. Esta decisión fue tomada para reducir el uso de área de FPGA y para acortar el tiempo de desarrollo. Por este motivo fue necesario escribir las rutinas de biblioteca que los controlan.

La implementación de la CPU y sus periféricos fue realizada en VHDL. Las herramientas de síntesis de *software libre* solo soportan Verilog, por lo que luego se migró a Verilog. Actualmente se encuentra disponible en ambos lenguajes.

Los mejores resultados de síntesis se obtuvieron utilizando *software libre* ya que estas herramientas permitieron aprovechar el 100 % de la FPGA. El sintetizador usado fue Yosys [12], que a su vez utiliza el Berkeley ABC [13]. El mismo logró optimizaciones tan buenas como el Synplify Pro incluido en las herramientas de Lattice.

Las características de la CPU implementada son:

- Memoria de programa: 7536 bytes (8192 total menos 656 para el *bootloader*, comparable a placas Arduino basadas en ATmega8)
- Memoria de datos: 512 bytes
- Frecuencia de reloj: 24 MHz (performance similar a un Arduino de 16 MHz)
- Entradas/salidas digitales: 15 (14 generales + 1 LED)
- Entradas analógicas: 8 (2 más que el UNO)
- Salidas analógicas (PWMs): 6
- UARTs: 1
- SPIs: 1 (hasta 24 MHz SCK)
- Timers independientes para generar tonos: 1
- Entradas de interrupción: 2
- Ocupación de la FPGA: 30 % aprox. (usando *software libre*, el doble usando las herramientas del fabricante)
- Tensión de I/O: 3,3 V

El conjunto resultante puede usarse fácilmente desde la interfaz de usuario de Arduino. Para lograr una mejor integración se adaptó el código del *bootloader* de los Arduinos basados en ATmega8 y se implementó el circuito de *auto reset* de Arduino.

V. Resultados

Se obtuvieron placas de fácil fabricación ya que ambas son de solo dos caras. Las separación entre pistas se fijó en 8 mils

y el ancho mínimo de pista en 10 mils. Estos valores fueron necesarios para poder resolver el conexionado de los circuitos integrados de paso fino (0,5 mm). Ambas placas pudieron ser soldadas manualmente.

MILK cumplió con su objetivo de aplicaciones generales ya que fue exitósamente utilizada para otras tareas tales como programar AVRs, depurar ARMs y configurar FPGAs de Xilinx.

El diseño de **Kéfir** permitió implementar un Arduino UNO, aunque con restricciones en la cantidad de memoria libre. Se pudo utilizar exitosamente un *shield* de Arduino con pantalla color y sensor táctil. No fue necesario ningún cambio circuital en el *shield*, ni en **Kéfir**. Sin embargo fue necesario trabajar un poco con las bibliotecas de soporte del display para evitar que rutinas que no se usaban consumieran memoria. En la Fig. 3 se observa el conjunto **Kéfir** y **MILK** junto al *shield* de Arduino utilizado.

Figura 3. Utilización de un *shield* de Arduino.

En el caso de Lattuino el área de FPGA utilizada es menor a una tercera parte, por lo que queda espacio más que suficiente para la implementación de periféricos adicionales.

VI. Conclusiones

Las herramientas de *software libre* mostraron ser adecuadas. En cuanto al diseño de los circuitos KiCad permitió realizarlo sin problemas. Las herramientas de síntesis (Yosys [12], Berkeley ABC [13], IceStorm [14] y Arachne PNR [15]) tienen algunas limitaciones, como por ejemplo no soportar VHDL, pero los resultados son comparables a los de las herramientas propietarias que ofrece Lattice. La gran ventaja de las herramientas de *software libre* en este caso fue la de poder usar toda el área de la FPGA.

La separación en dos placas, una con la FPGA y la otra para configuración, permitió reusar una parte importate del diseño para otras tareas de desarrollo.

La inclusión de conectores compatibles con Arduino permitió utilizar hardware ampliamente disponible.

Referencias

[1] S. E. Tropea, "Kéfir (Kit Educativo con FPGA, Inclusivo y Reciclable)," http://fpgalibre.sourceforge.net/Kefir/index.html.

[2] ——, "MILK (Multiple Interface Light Kit)," http://fpgalibre.sourceforge.net/Kefir/index.html.

[3] M. Banzi *et al.*, "Arduino UNO rev 3," https://store.arduino.cc/usa/arduino-uno-rev3.

[4] S. E. Tropea and R. A. Melo, "USB framework - IP core and related software," in *XV Workshop Iberchip*, vol. 1, Buenos Aires, 2009, pp. 309–313.

[5] J.-P. Charras *et al.*, "KiCad EDA: A cross platform and open source electronics design automation suite," http://www.kicad-pcb.org/.

[6] FTDI, "FT2232H - Hi-Speed Dual USB UART/FIFO IC," https://www.ftdichip.com/Products/ICs/FT2232H.htm.

[7] K. Waschk *et al.*, "Universal JTAG," http://urjtag.org/.

[8] D. Rath *et al.*, "Open on-chip debugger," http://openocd.org/.

[9] S. R. Ollie Lho *et al.*, "flashrom," https://www.flashrom.org/Flashrom.

[10] B. S. Dean *et al.*, "AVR Downloader/UploaDEr," https://-www.nongnu.org/avrdude/.

[11] S. E. Tropea and D. M. Caruso, "Microcontrolador compatible con AVR, interfaz de depuración y bus wishbone," in *Proceedings of the FPGA Designer Forum 2010*, Ipojuca, Brazil, 2010, pp. 1–6.

[12] C. Wolf, "Yosys Open SYnthesis Suite," http://www.clifford.at/yosys/.

[13] U. of California Berkeley, "ABC: A system for sequential synthesis and verification," https://people.eecs.berkeley.edu/ alanmi/abc/.

[14] C. Wolf, "Project IceStorm," http://www.clifford.at/icestorm/.

[15] C. Seed, "Arachne-PNR: Place and route tool for FPGAs," https://github.com/YosysHQ/arachne-pnr.

Repositorios abiertos para desarrollo con FPGAs

Rodrigo A. Melo, Bruno Valinoti
Instituto Nacional de Tecnología Industrial
Centro de Micro y Nanoelectrónica
Email: {rmelo, valinoti}@inti.gob.ar

***Resumen*—Los repositorios abiertos son una práctica generalizada en los proyectos tecnológicos, en búsqueda de compartir conocimiento y beneficiarse de los aportes de una comunidad. En este trabajo se presenta el estado actual de tres repositorios para el desarrollo con FPGAs, resultado de años participando en diversos proyectos, distribuidos bajo licencias libres: una biblioteca principalmente compuesta por código HDL, que incluye pequeños programas útiles para testeo y validación; aplicaciones para usar las herramientas provistas por los fabricantes de manera abstracta e independiente; ejemplos listos para usar que ejercitan características de distintas placas de desarrollo. Se describen sus características principales, el soporte brindado en la actualidad y cuáles son las mejoras planificadas como trabajo futuro. Los tres repositorios aumentan su contenido y material a través de las tareas cotidianas realizadas por los integrantes del grupo de desarrollo y están abiertos para la realización de Practicas Profesionales Supervisadas.**

***Index Terms*—FPGA, VHDL, Tcl, Python, Software Libre.**

I. Introducción

El uso de sistemas de control de versiones es una práctica ampliamente adoptada y difundida en el desarrollo de *software* y *hardware*. En la actualidad es común que empresas tecnológicas liberen y mantengan algunos de sus trabajos en plataformas con repositorios abiertos. La ventaja potencial es la creación de una comunidad alrededor, que los utilice activamente, reporte errores e incluso aporte mejoras al código. Git [1] es uno de los sistemas de control de versiones preferido para su implementación y GitHub [2] uno de los principales proveedores del servicio.

El grupo de desarrollo con FPGAs del Centro de Micro y Nanoelectrónica del Instituto Nacional de Tecnología Industrial (INTI), cuenta con años de trayectoria, durante los cuáles ha acumulado experiencia, creando código reutilizable y herramientas de desarrollo auxiliares.

En este trabajo se presenta el estado actual de los repositorios FPGA Lib [3], FPGA Helpers [4] y FPGA Examples [5], que fueron previamente presentados en [6]. Los mismos cuentan con recursos para el desarrollo con FPGAs y se distribuyen con licencias GPL 3 [7] o BSD de 3 cláusulas [8], están documentados en idioma Inglés y fueron desarrollados sobre sistemas Debian [9] GNU/Linux.

La estructura de este trabajo es la siguiente: Las secciones II, III y IV, presentan características y estado actual de los repositorios. Las secciones V y VI contienen resultados y conclusiones. Finalmente, la sección VII presenta el trabajo a futuro planificado.

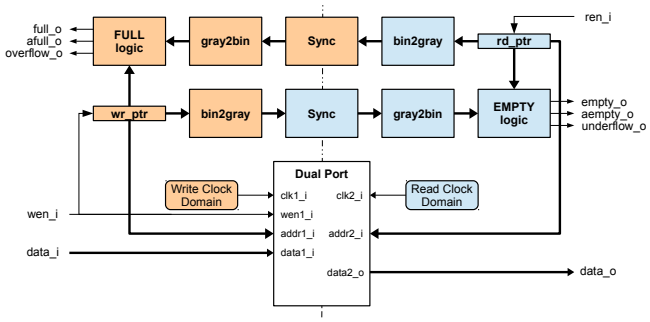


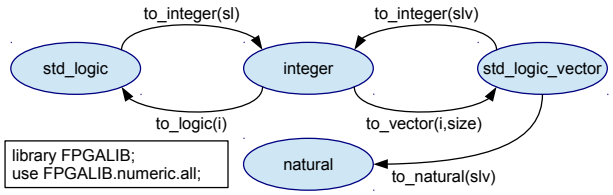
Figura 1. Diagrama en bloques de la FIFO [a]sincrónica.

II. FPGA Lib

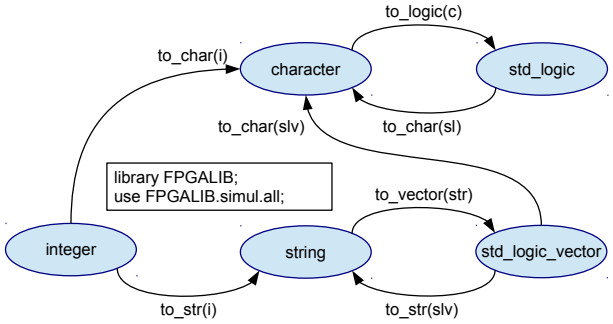
La participación en distintos desarrollos a lo largo del tiempo converge en la reutilización de código debido a que muchos diseños suelen compartir bloques o unidades funcionales. Para una integración más rápida a nuevos proyecto, facilidad de mantención de código y robustez, suele ser buena práctica la implementación de una biblioteca compartida. FPGA Lib es la colección de código HDL y pequeños programas auxiliares que se utilizan en los desarrollos del grupo de trabajo.

El repositorio se compone principalmente de código VHDL, con pequeños bloques, procedimientos y funciones, organizados en paquetes:

- **mems:** descripción de memorias *Simple*, *Dual* y *True Dual Port*, que son inferidas por las herramientas de síntesis. Se agregó una FIFO que puede ser usada tanto en modo sincrónico como asincrónico (Fig. 1).
- **numeric:** compuesto principalmente por funciones de conversión entre enteros y *std_logic_vectors* no cubiertas por los paquetes *std_logic_1164* y *numeric_std* (Fig. 2(a)). Adicionalmente, cuenta con funciones para obtener logaritmo, mínimo y máximo de enteros, las cuáles resultan particularmente útiles para trabajar con valores de *generic*s. Se agregó un contador y funciones para convertir entre código Binario Natural y Gray.
- **simul:** código útil para la realización de *testbench*es, como un generador de *clock* y *reset*, funciones de conversión entre *strings* y otros tipos (ver Fig. 2(b)), lectura y escritura de archivos con valores *std_logic* e impresión de mensajes en pantalla.
- **sync:** cadenas de *flip-flop*s, divisores de frecuencia de trabajo para generación de habilitaciones y bloques que



(a) Conversiones provistas por Numeric.



(b) Conversiones provistas por Simul.

Figura 2. Conversión entre tipos de datos que agregan los paquetes Numeric y Simul de FPGA Lib.

ayudan con la sincronización entre dominios de reloj (se agregó uno que utiliza codificación Gray).
- **verif:** bloques útiles a la hora de verificar diseños en *hardware*, como un parpadeo para *led*s, o un generador y comparador de valores para probar datos transmitidos y recibidos.

Al ser presentado en un trabajo previo, se discutió la posibilidad de agregar versiones en Verilog. Mantener código en más de un lenguaje presenta los desafíos de trabajo duplicado y posibles divergencias. Se realizaron pruebas sobre una de las ramas de desarrollo utilizando vhd2vl [10], pero aún con aportes al proyecto no se llegaron a realizar todas la conversiones necesarias. Actualmente se está desarrollando una nueva herramienta, denominada vhdl2verilog [11].

En cuanto a los programas auxiliares, se cuenta con varios *script*s Python, como por ejemplo: generador de números aleatorios, útil para estimular bancos de prueba; conversor entre números binarios y hexadecimales, que suelen utilizarse junto a una creador de ROMs; un realizador de gráficos a partir de un archivo con una o más columnas de datos.

Finalmente, se cuenta con un *Makefile* para GHDL [12] y algunos archivos de definición de pines para distintas placas.

FPGA Lib se distribuye bajo licencia BSD de 3 cláusulas.

III. FPGA Helpers

Este repositorio ofrece utilidades para manejar las herramientas de desarrollo para FPGAs en un modo independiente del fabricante. Está dividido en dos partes:

1. Archivos de *script*s *Tool Command Language* (Tcl) para realizar síntesis y programación, así como también un Makefile para facilitar la ejecución de los mismos.
2. Programas escritos en Python que ayudan a generar archivos de opciones y ejecutar los *script*s Tcl en forma efectiva para distintas tareas.

Los objetivos principales son:

- Facilitar la integración con sistemas de control de versiones, ya que los proyectos que generan las herramientas de los fabricantes son sumamente complicados de llevar en los mismos.
- No depender de las particularidades de las herramientas, dado que se modifica un único archivo que presenta siempre los mismos comandos y variables a asignar.
- Obtener reproducibilidad y repetibilidad, no dependiendo de la secuencia de pasos a realizar en una interfaz gráfica.
- Consumir menos recursos del sistema, lo cuál se obtiene evitando la interfaz gráfica, corriendo en una consola del sistema (particularmente útil cuando se ejecuta en dispositivos remotos).

Un desarrollo que utiliza FPGA Helpers necesita una única copia de *script*s Tcl, y luego por cada proyecto un archivo de opciones y su Makefile asociado. Los archivos base compartidos por todo el desarrollo son:

- **synthesis.tcl:** realiza la síntesis, implementación y generación de *bitstream*s. Permite seleccionar optimizaciones por área, velocidad y consumo de energía.
- **programming.tcl:** permite transferir *bitstream*s a dispositivos FPGAs. Adicionalmente soporta algunos tipos de memorias.
- **Makefile:** ejecuta los archivos Tcl con el interprete correspondiente según el fabricante seleccionado.

Luego, los archivos particulares para cada proyecto serán:

- **options.tcl:** archivo con opciones, tales como datos de dispositivos y archivos involucrados.
- **Makefile:** incluye al Makefile base, especificando únicamente la herramienta del fabricante a utilizar.

Además de la funcionalidad básica descripta, ofrecen algunas características extras tales como:

- Si el *script* de síntesis encuentra un archivo de proyecto para la herramienta seleccionada, lo utiliza (permitiendo así crearlo y configurarlo con la GUI del fabricante) y en caso contrario lo crea a partir de *options.tcl*.
- En *options.tcl* pueden utilizarse funciones para abstraer las tareas típicas de selección de dispositivo y archivos del proyecto, evitando así la necesidad de conocer los comandos propios de cada herramienta para tal fin.
- La función para selección de dispositivo posee un parámetro para indicar en que herramienta es válido. Con esto se logra poder sintetizar el mismo proyecto para distintas herramientas sin encontrarnos con el problema de dispositivo no soportado.
- Si hacen falta directivas específicas de una herramienta, se tiene acceso en *options.tcl* a una variable que indica el nombre de la herramienta en ejecución, para evitar haciendo uso de un condicional, errores de comando no soportado.

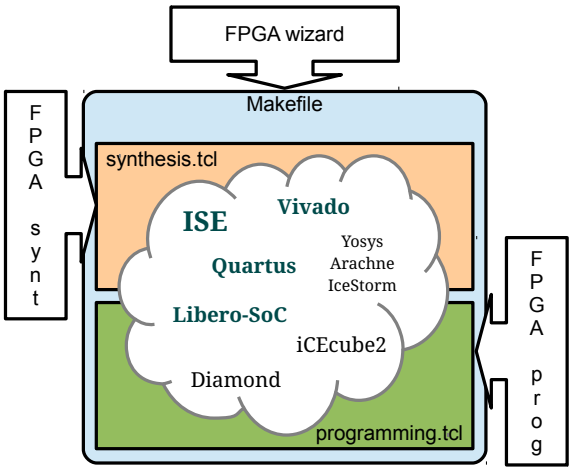


Figura 3. Esquema de relación entre scripts Tcl y Python

- El *Makefile* posee objetivos para abrir la GUI de la herramienta o ejecutar su consola Tcl.

Los principales programas Python que facilitan el trabajo con los *script*s Tcl son:

- **fpga_wizard:** utilidad interactiva que ayuda a generar el archivo de opciones y su Makefile.
- **fpga_synt:** permite sintetizar desde consola un proyecto creado con la herramienta del fabricante.
- **fpga_prog:** permite programar un bitstream del que dispongamos, sin necesidad de crearle un proyecto.

En la Fig. 3 se puede apreciar la relación entre archivos Tcl y que programa Python los utiliza.

FPGA Helpers se distribuye bajo licencia GPL 3.

IV. FPGA Examples

Cuando el grupo de trabajo adquiere una nueva placa o al investigar el uso de bloques en *hardware* para utilizar en un desarrollo, es común realizar un ejemplo. Este repositorio los recolecta y ordena. Sirve entre otras cosas para:

- Tener ejemplos que funcionan, listos para probar.
- Aprender a utilizar características de las FPGA o el *kit*.
- Tener un punto de partida para nuevos diseños.
- Concentrar material de consulta y aprendizaje.

Incluye a FPGA Helpers como submódulo para realizar la síntesis, implementación y generación de bitstream de forma automatizada, aunque alternativamente puede utilizarse la GUI del fabricante para armar un proyecto que incluya los archivos HDL y de *constraints*. Incluye también a FPGA Lib debido a que se utilizan algunos de sus bloques.

Por cada *kit* soportado, se cuenta con un directorio que incluye:

- Un *README* que detalla características de la placa (que contiene, cómo alimentar la placa y preparación del *hardware* para la programación).
- Un directorio (opcional) denominado *resources* que cuenta con lo necesario para generar bloques del fabricante (para evitar posibles problemas de licencia).
- Un directorio por cada ejemplo, utilizando nombres como gpios, clock, ddr, dma, axi, etc (agregando sufijos 2, 3, etc, cuando hay más de un ejemplo).

Cada directorio de ejemplo incluye:

- *README* con instrucciones para reproducirlo.
- Archivo *top.vhdl* del proyecto y opcionalmente *wrapper.tcl* (si hace falta simplificar el uso de un *core* del fabricante).
- Archivo de *constraints* con al menos asignación de pines.
- Archivos *options.tcl* y *Makefile* cuando se utiliza con FPGA Helpers.
- Para ejemplos complejos que lo ameriten, un directorio llamado *testbench* con *script*s e instrucciones para realizar una simulación.

Hay ejemplos básicos, disponibles para todos los *kit*s soportados, que muestran como trabajar con distintas fuentes de reloj y con las entradas/salidas disponibles, como *led*s, *switch*es y *dip-switch*es. Hay ejemplos avanzados para algunos *kit*s, que hacen uso de la DDR y DMA (un *loopback* sobre AXI DMA que se puede apreciar en la Fig. 4(a), y la realización de maestros y esclavos AXI propios a utilizar sobre las distintas interfaces del PS en dispositivos Zynq) y de *Giga Bit Transceiver* (Fig. 4(b)).

FPGA Examples se distribuye bajo licencia BSD de 3 cláusulas.

V. Resultados

El código VHDL de FPGA Lib está verificado y validado en *hardware*, dado que es utilizado activamente en los desarrollos del laboratorio. Los *script*s Python son utilizados con frecuencia, con lo cuál están depurados de errores básicos.

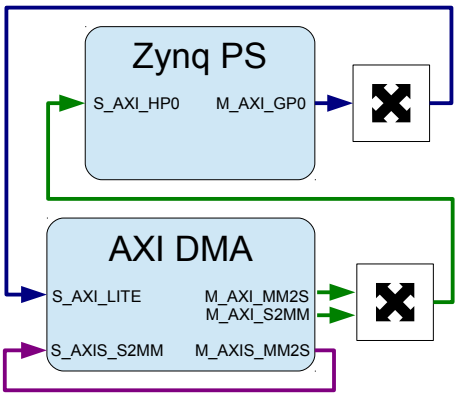
Los *script*s Tcl y programas de FPGA Helpers, soportan actualmente las herramientas Vivado e ISE de Xilinx, Quartus 2/Prime de Intel/Altera y Libero-SoC de Microsemi. En todos los casos se puede generar *bitstream* y programar la FPGA, pero para el caso particular de ISE, se puede además programar memorias SPI y BPI. Posee guía de usuario en inglés y castellano.

FPGA Examples provee aproximadamente 30 ejemplos distribuidos en 14 *kit*s que utilizan FPGAs de los cuatro proveedores principales:

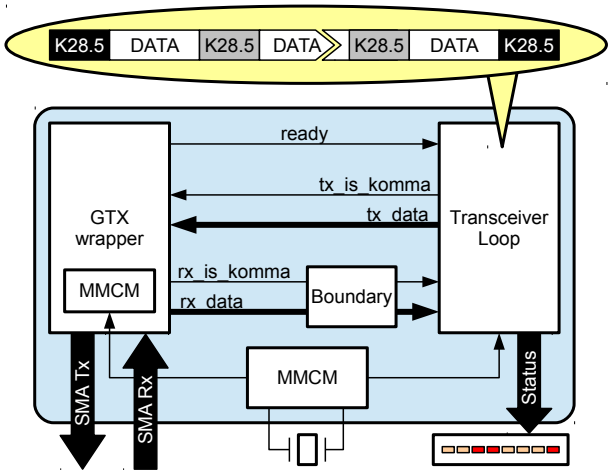
- **Xilinx:** Spartan-3 Development Kit, Avnet Spartan-6 FPGA LX9 MicroBoard, Spartan-6 FPGA SP601 Evaluation Kit, Virtex-6 FPGA ML605 Evaluation Kit, Digilent Zybo/ZedBoard/Pynq, ZC706 Evaluation Board y la CIAA ACC [13].
- **Intel/Altera:** DE0-Nano Development Board y MAX 10 FPGA (10M08) Evaluation Kit.
- **Microsemi:** M2S090TS-EVAL-KIT.
- **Lattice:** iCEstick Evaluation Kit y iCE40-HX8K Breakout Board.

VI. Conclusiones

A partir del trabajo diario de los integrantes del grupo de trabajo, se actualizaron los tres repositorios libres para el



(a) Ejemplo de loopback con una core AXI DMA.



(b) Ejemplo de loopback con un GigaBit Transciver.

Figura 4. Algunos de los ejemplos más avanzados incluidos en FPGA Helpers.

desarrollo con FPGAs, presentados en un evento previo. Los aportes más importantes fueron: nuevos componentes/funciones VHDL y *script*s Python en FPGA Lib; documentación en inglés y castellano de FPGA Helpers; agregados de varios ejemplos y placas en FPGA Examples, principalmente sobre el uso de DMA e interfaces AXI en dispositivos Zynq.

Por necesidades en FPGA Lib, se aportó al proyecto vhd2vl y se creo el proyecto vhdl2verilog, que actualmente compila de VHDL'93 a Verilog 2001 descripciones estructurales o combinacionales.

El ejemplo AXI DMA se portó fácilmente a la placa ZedBoard y se incluyó como practica en el *Advanced Workshop on FPGA-based Systems-On-Chip for Scientific Instrumentation and Reconfigurable Computing | (smr3249)* organizado por *The Abdus Salam International Centre for Theoretical Physics* (ICTP). Además, FPGA Lib está siendo utilizada en desarrollos del *Multidiciplinary LABoratory* (MLAB) de dicha institución.

Entre los ejemplos más destacados es menester nombrar el desarrollo de un maestro AXI en FPGA Examples, que se utiliza con las interfaces esclavas GP, ACP y HP de dispositivos Zynq-7000, dado que no se encuentra documentación al respecto y en su lugar hay abundante información sobre el *core* AXI DMA.

Los *script*s Tcl de FPGA Helpers deberían funcionar sobre cualquier SO, dado que dependen de las herramientas de los fabricantes y fueron desarrollados pensando en portabilidad entre las mismas, mientras que los *script*s de Python deberían funcionar si existe el interprete. En el caso de FPGA Examples, no debería haber problemas según el SO, pero en caso de haberlos, alcanza con tomar y utilizar los archivos HDL y de *constraints* en la herramienta del fabricante.

VII. Trabajo futuro

FPGA Lib crece continuamente a partir de proyectos con FPGAs y se espera agregar versiones Verilog del código a partir del trabajo activo sobre vhdl2verilog.

Se espera agregar a la brevedad soporte en FPGA Helpers de al menos las herramientas de Lattice Semiconductor y flujos de trabajo con herramientas Libres como Yosys [14], Arachne [15] y IceStorm [16].

En FPGA Examples se esperan ejemplos para nuevas placas de desarrollo, para otros bloques o características disponibles, y ejemplos más avanzados dentro de los *kit*s ya incluidos.

Referencias

[1] L. Torvalds, J. Hamano *et al.* Git. [Online]. Available: https://git-scm.com
[2] Github. [Online]. Available: https://github.com
[3] INTI CMNB. FPGA Lib. [Online]. Available: https://github.com/INTI-CMNB-FPGA/fpga_lib
[4] ——. FPGA Helpers. [Online]. Available: https://github.com/INTI-CMNB-FPGA/fpga_helpers
[5] ——. FPGA Examples. [Online]. Available: https://github.com/INTI-CMNB-FPGA/fpga_examples
[6] R. A. Melo and B. Valinoti, "Repositorios abiertos para desarrollo con FPGAs," in *Congreso Argentino de Sistemas Embebidos*, 2017, pp. 107–111.
[7] Free Software Foundation, Inc. GNU GENERAL PUBLIC LICENSE version 3. [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html
[8] University of California. The 3-Clause BSD License. [Online]. Available: https://opensource.org/licenses/BSD-3-Clause
[9] Debian Project. Debian: the universal operating system. [Online]. Available: http://www.debian.org
[10] Vincenzo Liguori, Larry Doolittle. VHD2VL. [Online]. Available: https://github.com/ldoolitt/vhd2vl
[11] Rodrigo A. Melo. vhdl2verilog. [Online]. Available: https://gitlab.com/-rodrigomelo9/vhdl2verilog
[12] T. Gingold. Where VHDL meets gcc. [Online]. Available: http://ghdl.free.fr/
[13] N. S. S. Bruno Valinoti, Rodrigo A. Melo and D. F. Alamon, "Computadora Industrial Abierta Argentina para aplicaciones de Alta Capacidad de Cómputo," in *A presentarse en el Designer Forum del SPL2019*, April 2019.
[14] C. Wolf. Yosys Open Synthesis suite. [Online]. Available: http://www.clifford.at/yosys
[15] Arachne: Place and route tool for FPGAs. [Online]. Available: https://github.com/cseed/arachne-pnr
[16] C. Wolf. Project IceStorm. [Online]. Available: http://www.clifford.at/-icestorm/

Plataforma Digital de Bajo Costo para Procesamiento de Señales Ultra-Wideband

Edgardo Marchi, Marcos Cervetto and Pablo Gámez
Instituto Nacional de Tecnología Industrial (INTI), Argentina
Centro de Electrónica e Informática
emails: emarchi,cervetto,pgamez@inti.gob.ar

***Resumen*—Se presenta el diseño y la implementación de una plataforma digital para señales impulsivas UWB, utilizando componentes off-the-shelf de bajo costo y la placa de desarrollo Open Hardware CIAA-ACC, con el propósito de probar algoritmos de procesamiento de señal sobre la misma. Para eso, se concibe una arquitectura modular sencilla. En el presente trabajo se elabora con cierto detalle sobre los pasos principales del proceso de diseño y se discuten las alternativas, las decisiones tomadas, sus ventajas y desventajas.**

***Index Terms*—Wireless communications, UWB, ultra-wideband, testbed.**

I. Introducción

Los sistemas ultra-wideband (UWB) son, hablando en términos generales, sistemas de radio en los cuales el ancho de banda instantáneo es como mínimo 500MHz o más del 20 % de la frecuencia central. En la actualidad, los sistemas UWB se han convertido en una tecnología emergente debido a sus características únicas que hacen viable aplicaciones en diversos campos de las radiocomunicaciones. Estas aplicaciones incluyen posicionamiento [1], detección de objetos ocultos [2], imágenes, monitoreo y diagnóstico médico [3], [4], comunicaciones de corto alcance [5], entre otras. Muchas aplicaciones utilizan radios impulsivas, esto es, sistemas que transmiten y reciben pulsos muy cortos en tiempo [6].

Durante el estudio y desarrollo de algoritmos y técnicas para procesamiento de señales, una plataforma de hardware digital versátil se vuelve indispensable para implementar y verificar su correcto funcionamiento. Hay un número de soluciones comerciales en este aspecto para UWB, pero su costo sigue siendo demasiado alto en la mayoría de los casos. Además, muchas poseen limitaciones en cuanto al acceso a los componentes de hardware y software del sistema, lo cual dificulta la validación de los algoritmos diseñados. Algunas están apuntadas a aplicaciones específicas, como localización [7], [8] o detección [9]. Otras son onerosas [8], [9] o de especificaciones limitadas [7], [8].

Además del bajo costo, las características principales de la plataforma digital desarrollada son el sistema de muestreo en tiempo equivalente y la poca cantidad de recursos digitales utilizados, brindando área de FPGA y tiempo de CPU para la implementación y prueba de algoritmos. Además, fue desarrollado un sistema en tiempo real de adquisición y graficación de señales.

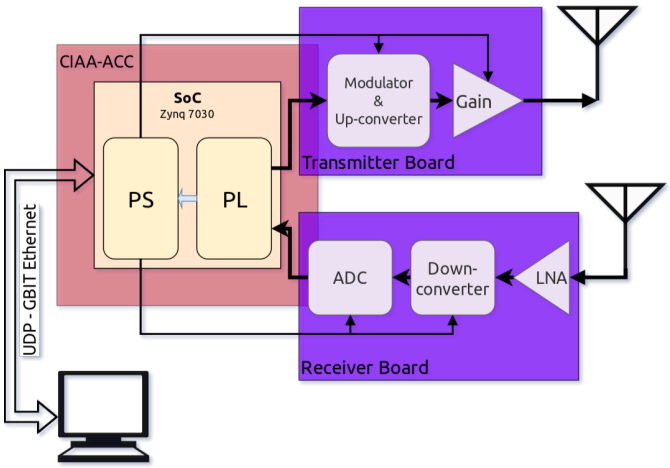


Figura 1. Diagrama de la plataforma.

II. Descripción de la arquitectura

Una descripción completa de la arquitectura puede ser encontrada en [10] incluyendo el front-end analógico. Los bloques principales de la plataforma completa son mostrados en Fig. 1. La misma cuenta con una placa central de procesamiento digital open source (CIAA-ACC [11]) y dos placas de front-end analógico, una transmisora y otra receptora.

La unidad central de procesamiento utiliza un SoC Zynq-7030. Al combinar área de FPGA (en adelante PL según nomenclatura de Xilinx) con un procesador dual-core ARM Cortex-A9 (en adelante PS), posee las capacidades para implementar algoritmos de procesamiento en tiempo real tanto en hardware como en software. El SoC es responsable de cuatro tareas principales: la generación del pulso, la reconstrucción de la señal, la comunicación con un dispositivo de visualización y post-procesamiento (una PC, por ejemplo) y la configuración de los periféricos de la plataforma.

II-A. Arquitectura implementada en la PL

Un diagrama de la arquitectura de hardware implementada en la PL se puede ver en la Fig. 2.

Circuito generador de pulsos: Como fue mencionado en la sección anterior, los pulsos son generados con la PL del Zynq Z7030 de Xilinx. Este SoC permite la generación de pulsos

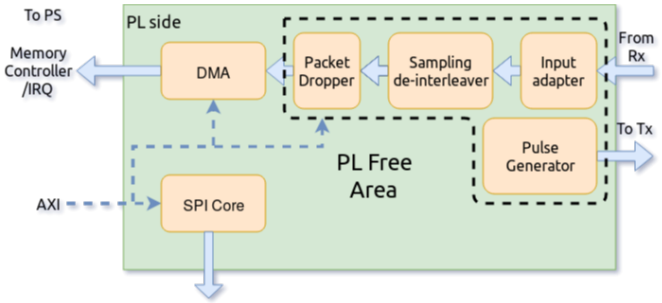


Figura 2. Arquitectura implementada en el PL

tan cortos como 1.25ns [12] y tiene salidas con impedancia controlada digitalmente, lo que permite una conexión directa entre el chip y el modulador. El pulso es generado simplemente por un contador que produce un $'1'$ durante un solo período de clock cuando alcanza su máxima cuenta. Debido a restricciones del diseño analógico[10], este módulo puede funcionar hasta 504 MHz (obtenidos como $f_{RS} * 63/10$ siendo f_{RS} la frecuencia real de muestreo según sec. A), lo que produce un pulso con un ancho de banda de 1008 MHz. La cuenta máxima del contador se ajusta de acuerdo a la frecuencia de repetición (P_{RF}) de la señal, en este caso, con una P_{RF} = 7,875MHz resulta una cuenta máxima de 64.

Reconstrucción de señal: La reconstrucción de la señal es llevada a cabo en dos etapas. Primero, se requiere de una adaptación de los datos de entrada ya que el ADC entrega las muestras en una interfaz serie de dos líneas para cada canal (I/Q)[13]. Esta adaptación implica no sólo la conversión SIPO, sino también la conversión de clock del ADC al clock del sistema (80MHz). Cabe mencionar que ambos clocks son sincrónicos. El bloque de la interfaz de adaptación se muestra en la Fig. 3.

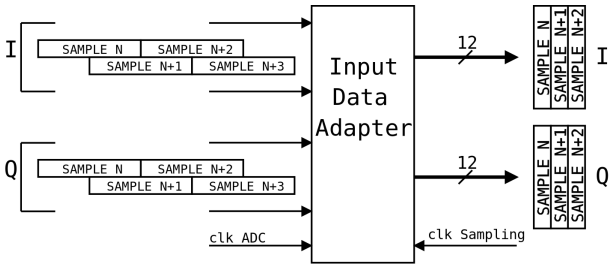


Figura 3. Bloque adaptador de entrada

Los recursos de PL usados por este bloque son presentados en el cuadro I.

Resource	Quantity
LUTs	29
Registers	130

Cuadro I
RECURSOS DE PL USADOS POR EL ADAPTADOR DE ENTRADA

La segunda etapa de la reconstrucción de la señal involucra el desentrelazado de las muestras (tomadas a f_{RS}) para formar la señal recibida muestreada a f_{EQ}. Este proceso es resuelto por un buffer ping-pong direccionado por una lógica K-modular, que consiste en una dirección de escritura generada por un contador de a pasos en módulo K. Por ejemplo, en la configuración usada en este prototipo, las posiciones de la memoria de almacenamiento serían $\{0, 63, 126, .., 630, 53, 116, .., 620, 43, 106, ..\}$. Entonces, si se leen secuencialmente, el efecto de entrelazado del muestreo en tiempo equivalente es cancelado y una copia de la señal es obtenida. La Fig. 4 muestra un diagrama de esta etapa

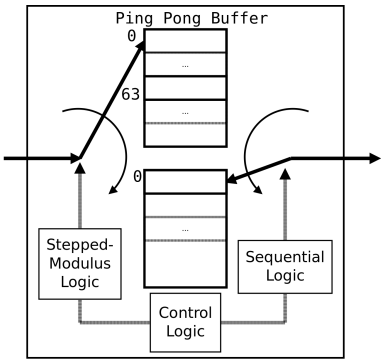


Figura 4. FPGA de-interleaving block

El módulo es instanciado para ambos canal I y canal Q. Cada canal requiere 2×640 posiciones de memoria de 12 bits cada una, produciendo un total de 15360 bits de memoria por canal, que pueden ser almacenados en una sola BlockRAM. [14]. Luego de reconstruir la señal, las muestras son escritas en la memoria DDR principal de la plataforma a través de un bloque AXI DMA conectado al puerto HP0 del PS.

Control de flujo: Dada la elevada tasa de datos, se implementó un módulo que descarta un cierto número configurable de paquetes. Esto produce por un lado que el enlace con la PC no se sobrecargue, pero sobre todo disminuye la carga del PS, ya que con cada paquete listo para transmitir a la PC, se interrumpe la ejecución del procesador.

II-B. Arquitectura de la aplicación en el PS

El PS del SoC es responsable de enviar los datos de la señal reconstruida a la PC y oficiar de interfaz entre el software de configuración (ver sec. II-C) y los dispositivos SPI. Ambas tareas son ejecutadas concurrentemente y manejadas por el sistema operativo FreeRTOS [15]. La transmisión de datos a la PC se ejecuta por medio de interrupciones del DMA, dónde con cada interrupción se le entrega un semáforo binario a la tarea encargada de convertir y enviar la trama. Los datos de la señal son enviados a través del protocolo UDP en una conexión confiable entre el SoC y la PC. De esta forma se aprovecha el encabezado acotado de UDP respecto de otros protocolos. Los paquetes UDP son enviados utilizando la biblioteca lwIP, a través del core Giga-bit Ethernet (GbE) incluido en el SoC. Por otro lado, la tarea encargada de programar los dispositivos SPI se ejecuta en un loop con baja prioridad. Un IP core SPI-master fue incluido en la

configuración del hardware a instanciar en la PL, de forma de programar el LTC5586 (demodulador), el ADC12DS105 (ADC), el TRF372017(modulador) y el PE43712A (atenuador programable).

II-C. PC software

En la PC, fue diseñado un sistema en tiempo real codificado en *C++* que permite acceder a las muestras a través de la interface de red, graficarlas y capturarlas en disco para posterior análisis. El software recibe las muestras por medio del link UDP/GbE mencionado en la sec. II-B. En la Fig. 5 se muestra una captura del software en cuestión. A su vez, escribe, lee y almacena la configuración de los diferentes dispositivos presentes en el receptor y transmisor, por medio de una interfaz SPI. La configuración se hace desde una interfaz de alto nivel, de manera que el valor hexadecimal de los registros es calculado automáticamente.

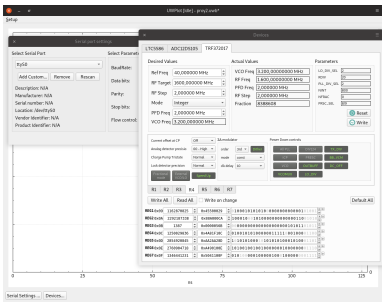


Figura 5. Software de captura, configuración y graficación del sistema

III. Construcción y Mediciones del Prototipo Completo

El transmisor y receptor (Fig. 6) fueron diseñados en en 4 capas de material Isola FR408 [16] ya que su permitividad varía entre 3.69 @ 0.1 GHz a 3.65 @ 10 GHz, lo que resulta excelente para este diseño.

En la Fig. 7 se muestra el pulso generado por la FPGA (PL).

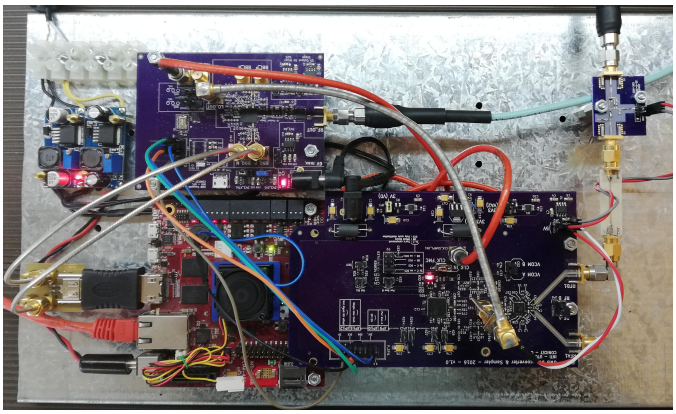


Figura 6. Hardware de la plataforma incluyendo transmisor y receptor junto a la CIAA-ACC.

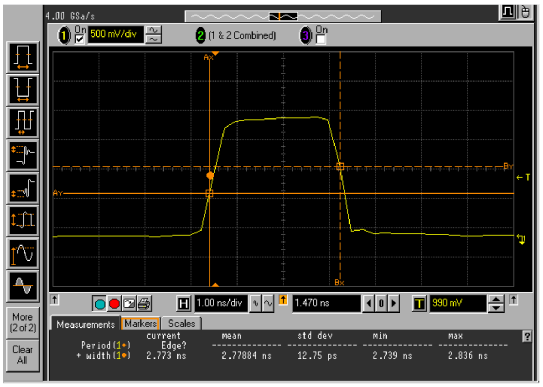


Figura 7. FPGA-generated pulse.

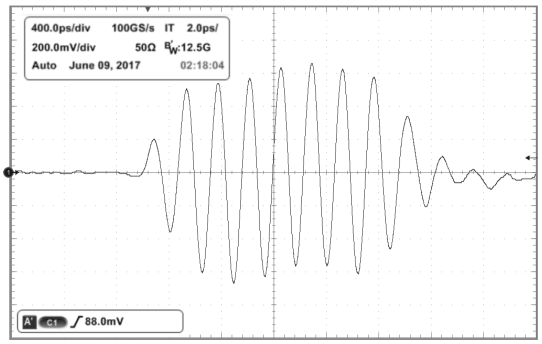


Figura 8. Pulso modulado a la salida de la placa del transmisor.

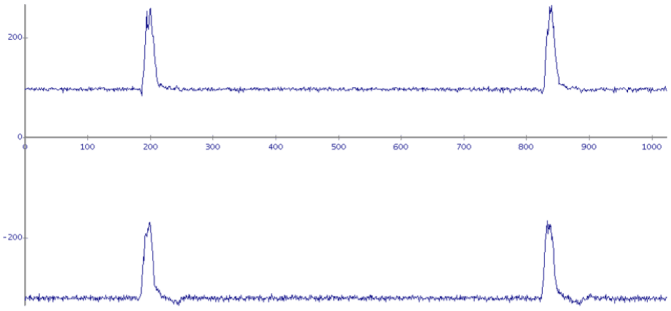


Figura 9. Pulso reconstruido (canales I y Q).

El pulso en banda pasante se puede ver en Fig. 8. Dicho pulso fue medido con un osciloscopio Tektronix MSO71254C de $BW = 12{,}5\ GHz$ de por medio de un cable SMA.

Finalmente el pulso reconstruido es mostrado en la PC, tal como se presenta en la Fig. 9. El setup completo de la plataforma se muestra en la Fig. 10.

IV. Conclusiones

Este trabajo presenta una arquitectura simple, de bajo costo de una plataforma para señales UWB impulsivas que puede ser usada para desarrollar y validar aplicaciones de sensado y detección. Aunque el primer prototipo fue testeado con un ancho de banda de 500 MHz, la arquitectura puede ser extendida fácilmente hasta 1 GHz.

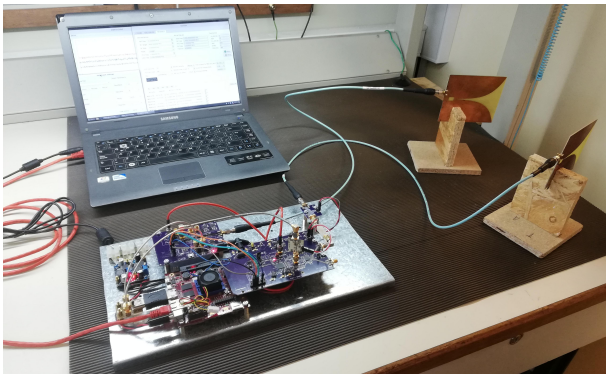


Figura 10. Banco de prueba de la plataforma UWB completo, incluyendo antenas y PC para visualización y post-procesamiento.

V. Trabajo futuro

Mediante diversidad espacial se puede lograr mejorar el rango de aplicaciones de UWB incluyendo, por ejemplo, el armado de imágenes. Para ello está planeado el procesamiento de varios canales en paralelo. También se planea implementar algoritmos de extracción de features de la señal sobre el SoC, como por ejemplo el *Matrix Pencil Method*[17].

Apéndice A
Esquema de muestreo

El gran ancho de banda de las señales UWB requiere una tasa de muestreo muy elevada. En este caso particular sería necesaria una frecuencia superior a 1 GSps, por lo tanto, dadas las restricciones en el costo, un muestreo de Nyquist no resulta viable. Utilizando la naturaleza periódica de la señal, surge como solución un esquema de muestreo en tiempo equivalente (ETS). En [18] se propone un método usando lineas de retardo digital (DDL) y un ADC de 500 MSps, y [19] usa cuatro ADCs de 1 GSps que son muy costosos. Los autores en [20] analizan distintas alternativas que van desde *time-interleaved ADC*, *hybrid filter bank*, *Xsampling* hasta *compressed sensing*. Se encontró muchas de estas opciones imprácticas, debido a que la señal recibida requiere un ADC con un ancho de banda analógico muy grande y las técnicas de ETS por otro lado utilizan frecuencias de muestreo muy bajas. Estos dos requerimientos contrapuestos no podían ser satisfechos con conversores comunes. Este trabajo adopta un método diferente y simple, llamado "sampleo híbrido con frecuencias fraccionales". Está basado en el hecho que al muestrear a una frecuencia que sea una fracción irreducible de la frecuencia de repetición de la señal de entrada, después de una cierta cantidad de períodos, se puede reconstruir enteramente la señal original. Es decir, dados dos enteros K y N tal que K/N es una fracción irreducible. Sean f_{RS} la frecuencia real de sampling y P_{RF} la frecuencia de repetición de la señal (o frecuencia de repetición de pulsos en este caso). Ahora, si la frecuencia equivalente f_{EQ}, es elegida para satisfacer la Eq. (1)

$$\begin{aligned} f_{RS} \cdot N &= f_{EQ} \\ P_{RF} \cdot K &= f_{EQ} \end{aligned} \tag{1}$$

entonces, la señal original se puede reconstruir después de juntar N periodos. Esta solución evita el uso de DDLs. En particular, en este diseño se eligió una $f_{RS} = 80\ MHz$, que permite seleccionar un ADC barato y de alta disponibilidad. Usando $K/N = 640/63$ y una $P_{RF} = 7{,}875$MHz (que puede ser generada por los MMCM) resulta en $f_{EQ} = 5{,}04$GHz. Finalmente para el caso de prueba se eligió $f_c = 480 \cdot \text{PRF} = 3{,}78$GHz.

Referencias

[1] S. Gezici, Zhi Tian, G. B. Giannakis, H. Kobayashi, A. F. Molisch, H. V. Poor, and Z. Sahinoglu. Localization via UWB radios: a look at positioning aspects for future sensor networks. *IEEE Signal Processing Magazine*, 22(4):70–84, July 2005.

[2] J. Li, L. Liu, Z. Zeng, and F. Liu. Advanced signal processing for vital sign extraction with applications in uwb radar detection of trapped victims in complex environments. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(3):783–791, March 2014.

[3] S. K. Davis, B. D. Van Veen, S. C. Hagness, and F. Kelcz. Breast tumor characterization based on ultrawideband microwave backscatter. *IEEE Transactions on Biomedical Engineering*, 55(1):237–246, Jan 2008.

[4] Mohammad Ghamari, Balazs Janko, R. Simon Sherratt, William Harwin, Robert Piechockic, and Cinna Soltanpur. A survey on wireless body area networks for eHealthcare systems in residential environments. *Sensors*, 16(6):831, 2016.

[5] Liuqing Yang and G. B. Giannakis. Ultra-wideband communications: an idea whose time has come. *IEEE Signal Processing Magazine*, 21(6):26–54, Nov 2004.

[6] M. Z. Win and R. A. Scholtz. Impulse radio: how it works. *IEEE Communications Letters*, 2(2):36–38, Feb 1998.

[7] A. De Angelis, M. Dionigi, A. Moschitta, and P. Carbone. A low-cost ultra-wideband indoor ranging system. *IEEE Transactions on Instrumentation and Measurement*, 58(12):3935–3942, Dec 2009.

[8] A. Feldman, A. Bahr, J. Colli-Vignarelli, S. Robert, C. Dehollain, and A. Martinoli. Toward the deployment of an ultra-wideband localization test bed. In *Vehicular Technology Conference (VTC Fall), 2011 IEEE*, pages 1–5, Sept 2011.

[9] Q. Liu, Y. Wang, and A. E. Fathy. A compact integrated 100 GS/s sampling module for UWB see through wall radar with fast refresh rate for dynamic real time imaging. In *Radio and Wireless Symposium (RWS), 2012 IEEE*, pages 59–62, Jan 2012.

[10] P. Gámez, E. Marchi, M. Cervetto, C. Giuffrida, G. Perez, A. Altieri, and C. Galarza. A low-cost ultra-wideband test-bed for dielectric target detection. In *2017 XVII Workshop on Information Processing and Control (RPIC)*, pages 1–6, Sep. 2017.

[11] http://www.proyecto-ciaa.com.ar/devwiki/doku.php?id=desarrollo:ciaa_acc:ciaa_acc_inicio.

[12] Zynq-7000 SoC: DC and AC Switching Characteristics - DS191.

[13] http://www.ti.com/product/adc12ds105.

[14] 7 Series FPGAs Memory ResourcesUser Guide - UG473.

[15] https://www.freertos.org.

[16] http://www.isola-group.com/products/all-printed-circuit-materials/fr408/.

[17] T. K. Sarkar and O. Pereira. Using the matrix pencil method to estimate the parameters of a sum of complex exponentials. *IEEE Antennas and Propagation Magazine*, 37(1):48–55, Feb 1995.

[18] Chao Chen, Shiyou Wu, Shengwei Meng, Jie Chen, Guangyou Fang, and Hejun Yin. Application of equivalent-time sampling combined with real-time sampling in uwb through-wall imaging radar. In *Instrumentation, Measurement, Computer, Communication and Control, 2011 First International Conference on*, pages 721–724. IEEE, 2011.

[19] Matthew Bruce Blanton. *An FPGA software-defined ultra wideband transceiver*. PhD thesis, Virginia Tech, 2006.

[20] Zhou Yan, Zhou Cong, and Wang DongH. Low rate sampling techniques for uwb systems: A survey. In *Control Conference (CCC), 2015 34th Chinese*, pages 7777–7782. IEEE, 2015.

Propuesta de solución de problema de Clock Domain Crossing en un IP Core AXI to AHB

1st Andrés J. Demski
FPGA Engineer
Satellogic S.A.
Buenos Aires, Argentina
ademski@satellogic.com

2nd David M. Caruso
FPGA Engineer
Satellogic S.A.
Buenos Aires, Argentina
david@satellogic.com

3rd Andrés Miguel Airabella
FPGA Engineer
Satellogic S.A.
Buenos Aires, Argentina
aairabella@satellogic.com

***Resumen*—Cuando se desea conectar un dispositivo AHB (Advanced High-performance Bus) a un bus AXI (Advanced eXtensiable Interface), es necesaria la utilización de un conversor. Este debe realizar las tareas de conversión de protocolo y adaptación de cruce de dominio de reloj (en caso de que los buses utilicen distinta fuente de reloj, con distinta fase y/o frecuencia). Originalmente, se utilizó el core propuesto por el fabricante de la FPGA (Field Programmable Gate Array) utilizada y se encontraron fallas en su funcionamiento. Además, el proceso de generación de bitstream demoraba aproximadamente una hora.**

En este trabajo se presenta una solución para la interfaz entre buses AHB y AXI, la cual es eficiente en recursos, mejora el tiempo de generación de bitstream y agrega confiabilidad del sistema.

***Palabras Clave*—AMBA, AXI, AHB, CDC, Microsemi, Reloj, FPGA**

I. Introducción

El desarrollo de *System on Chips* (SoC) implica la interconexión de distintos cores y periféricos mediantes buses. Para ello, ARM establece especificaciones bajo el nombre de *Advance Microcontroler Bus Architecture* (AMBA). La misma presenta varias versiones y definen interfaces de comunicación, entre ellas AXI (Advanced eXtensiable Interface) [1] y AHB (Advanced High-performance Bus) [2].

Si se desea conectar dispositivos AHB a un bus AXI, o viceversa, es necesario utilizar un core que resuelva la traducción entre protocolos y, además, debe resolver el cruce de dominio de reloj si es que los buses se encuentra bajo diferentes relojes. Los fabricantes de FPGA (Field Programmable Gate Array) suelen dar este conversor dentro de su catálogo de cores para poder hacerlo de forma transparente y poder utilizar todos los cores que tienen desarrollados sin que la interfaz que tengan sea un limitante.

Cuando un sistema se encuentra enteramente en un único dominio de relojes, las herramientas de síntesis y *Place and route* disponen las unidades lógicas de forma que se cumplan los tiempos de *hold* y *setup* de los flip-flops. El incumplimiento de estos tiempos puede provocar que la salida del flip-flop no se pueda determinar, provocando un fenómeno conocido como metaestabilidad [3]. Si se realiza el cruce de dominio de reloj, el cumplimiento de los tiempos de *hold* y *setup* no se puede asegurar y hay que realizar diseños que disminuya la probabilidad de ocurrencia de este fenómeno. Al método para enviar datos y señalizaciones de un dominio de reloj a otro se llama *Clock Domain Crossing* (CDC). Además, los métodos tradicionales de simulación y análisis estático de *timing* no son suficientes para validar un diseño y no garantizan que funcione en hardware para todas las condiciones.

Para afrontar problemas de CDC, se deben diseñar sistemas con señalización compatible con la relación de relojes que se tiene y utilizar sincronizadores para reducir la probabilidad de recibir datos erróneos. También, se deben estipular *constraints* (restricciones de diseño) a las herramientas de síntesis y ruteo, para especificar que señales hay que analizar para poder reducir el tiempo de procesamiento, evitar simplificación de lógica y obtener una implementación que sea siempre valida sin depender de la aleatoriedad del proceso.

II. Descripción del problema

En la Fig. 1 se puede visualizar un diagrama en bloques del sistema.

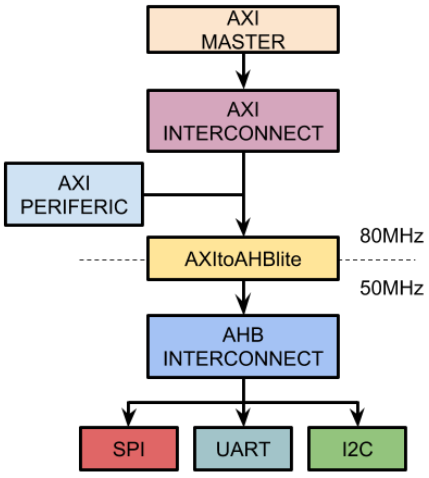


Fig. 1. Diagrama en bloques del sistema

El sistema esta compuesto por una CPU que funciona como AXI Master a 80MHz, encargada de iniciar las transacciones. Los bloques *interconnect* permiten interconectar varios periféricos a un mismo *master*. El core AXItoAHB realiza la conversión de AXI a AHB y el cruce de dominio de reloj de

80Mhz a 50Mhz. El core utilizado para realizar esta tarea es el AXItoAHBlite [4] del catálogo de *Microsemi*. En el bus AHB se encuentran periféricos de baja velocidad como pueden ser UART, SPI, I2C, entre otros.

Cuando se realizan transferencias de datos desde el AXI master hacia los periféricos ubicados del lado AHB, el bus AHB se bloquea de forma aleatoria, provocando que el AXI bus quede inhabilitado. Se pierde control de los periféricos en el bus AXI como AHB y la única forma de recuperarlos es reiniciando el sistema completo.

A. Búsqueda de la falla

Para descartar que fuese un problema lógico, se realizaron simulaciones intentando que sean lo más representativas de la realidad incluyendo corrimiento de los relojes. Con simulaciones que cubrían todos los casos de uso, se determinó que el problema no estaba relacionado con una mala descripción lógica. A su vez, comprobando que la configuración del proyecto era correcta, que la herramienta no señalaba problemas en el diseño y el software asociado hacia un correcto uso del hardware en cuestión, se abordó el ultimo aspecto que está relacionado a aspectos físicos ligados al timing del diseño

Utilizando 3 setups diferentes y probando un mismo bitstream en cada uno de ellos, se observó que la falla se producía de formas distinta en cada uno de ellos. En el setup de laboratorio, la probabilidad de falla era baja. En cambio, en sistemas que estaban a diferente temperatura y en un entorno eléctrico distinto, la falla se producía con mayor frecuencia o nunca funcionaba en el peor de los casos.

En los reportes del proceso de síntesis y *place and rounte* (PnR) se observó que la totalidad del proceso duraba 1 hora en promedio. Además, se obtenía un *worst slack* de -2.2 ns en las iteraciones y el *signal path* que lo provocaba estaba situado en el core AXItoAHB.

Utilizando los reportes del proceso de PnR, y relacionándolo con las pruebas en hardware de los bitstreams, se concluyó que el problema era de CDC. Se logró identificar que el problema se encontraba en el conversor AXI a AHB proporcionado por Microsemi.

III. Propuesta de solución

Inicialmente, se propuso intentar corregir el core AXItoAHBlite de Microsemi. Se observó en los archivos fuentes que internamente no resuelve de forma correcta los cruces de dominio de reloj. También se observó que el core de catálogo implementa más funciones de las que se necesitan para la aplicación actual. A su vez, la complejidad con la cual estaba escrito, no permitía realizar modificaciones de forma simple y en un tiempo acotado. Es por ello que se decidió realizar un core propio que resuelva el problema.

Se estipularon una serie de requerimientos a cumplir para el diseño del nuevo core:

- El reloj de AXI es menor que el doble de AHB. (AXI: 80MHz, AHB:50MHz).
- La escritura no requiere validación.

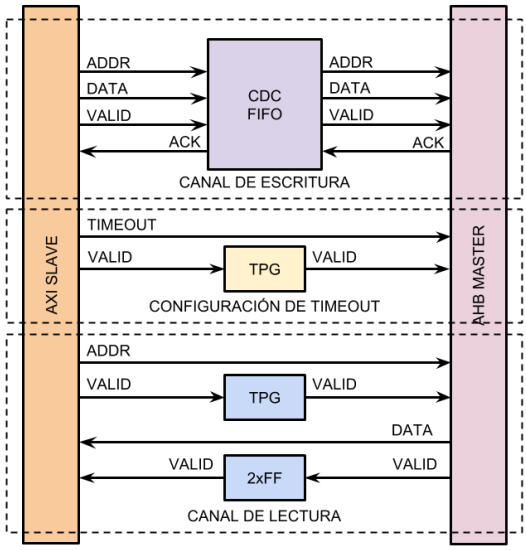


Fig. 2. Diagrama en bloques de la solución propuesta

- En caso de ingresar un ADDR inválido y no tener respuesta del lado AHB, terminar la transacción por timeout.
- Soporte de AHB lite (acceso a registros).
- No es necesario soportar BURST.
- Debe soportar lectura y escritura simultánea.

En la Fig. 2 se observa el diagrama en bloques de la solución propuesta. El problema fue divido en dos partes: 1) Resolver las comunicaciones. 2) Realizar el cruce de dominio de reloj.

Los bloques que figuran con el nombre de *AXI Slave* (El cual se conectaría con el *AXI Master* del sistema) y *AHB Master* (El cual se conecta a los periféricos que se desean controlar) se encargan de traducir las señalizaciones de los protocolos a un tipo de bus más simple para poder realizar el pasaje de dominio de reloj de forma más transparente y robusta.

Para resolver el cambio de dominio de reloj, se buscaron soluciones independientes tanto para lectura como para escritura. Además, se agregó una interfaz de configuración de timeout para hacer el sistema más robusto y que ante cualquier problema en el lado AHB, se libere el recurso AXI.

A. Canal de escritura

Considerando que el proceso es unidireccional y que no puede haber error en la escritura, se propuso usar una FIFO con señalización compatible para realizar CDC como se puede ver en la Fig. 3.

La FIFO está desarrollada con una memoria de doble acceso (Dual Port RAM), la cual cada puerto puede estar a relojes diferentes sin restricciones entre ellos. Las señalizaciones de disponibilidad de datos se realizan a través de la comparación de las direcciones de lectura y escritura, para lo cual se utilizaron los conversores de código *gray-binario* para reducir la probabilidad de la lectura de una dirección errónea como se explica en [5].

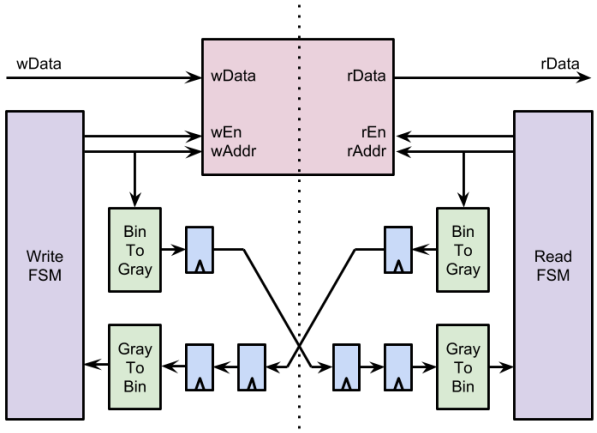


Fig. 3. Fifo CDC

B. Canal de lectura

Cuando se realiza una lectura, primero se envía la dirección de memoria a la que se desea leer y posteriormente se recibe el dato. Esto hay que realizarlo con dos métodos de sincronismo diferentes.

Para enviar la dirección a traves del cambio de dominio de reloj, se procedió a utilizar un bloque conocido como *toggle-pulse generator* (TPG) [5], el cual se muestra en la Fig. 4.

En la Fig. 5, se observa como responde el TPG a un pulso que dura un ciclo de reloj. El mismo, va a producir que la señal que debe realizar el cruce de dominios de reloj tenga la duración de 2 ciclos. Con esto, se garantiza que el flip-flop que esta bajo el otro dominio de reloj vea esta señal. Este diseño, utilizado de esta forma, requiere que el reloj de salida sea mayor que la mitad de el de entrada, de lo contrario, existe probabilidad de perder el dato.

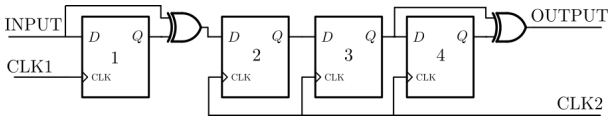


Fig. 4. TPG: Toggle-Pulse generator

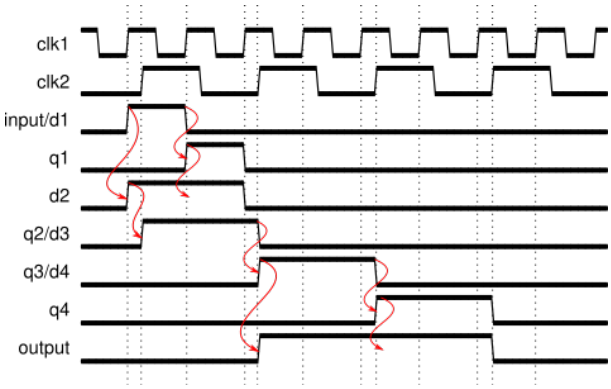


Fig. 5. Waveform del TPG

Para realizar la lectura del dato, se utilizó un método más simple, el cual se basa en la utilización de dos flip-flops bajo el dominio de reloj de AXI. En este caso, hace falta sincronizar únicamente la señalización ya que se considera en el diseño que el dato no va a cambiar hasta el próximo pedido de lectura.

C. Configuración de Timeout

Para realizar la configuración de timeout se utilizó el mismo tipo de sincronismo que el envío de la dirección en el canal de lectura. Esto se debe a que no se espera validación de esta configuración y se supone que siempre se efectúa correctamente.

IV. Validación de diseño

Las pruebas que se realizaron para validar el diseño fueron tanto en simulación como ejecutando sobre hardware. Se utilizaron los scripts diseñados para diagnosticar el problema y se desarrollaron nuevos que utilizaban varios periféricos AXI y AHB en simultáneo para aumentar la probabilidad de fallas. A pesar de los intentos que se realizaron, nunca se pudo volver a producir una falla y se validó que el diseño funciona.

Además, se observa una mejoría en el proceso de síntesis y PnR, reduciendo el tiempo del proceso de 1 hora a 25 minutos. Respecto al *worst slack*, se mejoró de -2.2ns a 0.2ns y ya el *signal path* que provoca el mismo no está en el core bajo análisis.

V. Conclusión

Con el diseño propuesto, se pudo lograr que el sistema sea confiable y reproducible. Se logró hacer que el proceso siempre concluya en una solución válida que funcione en todos los dispositivos a distintas condiciones de temperaturas. Con esto, se pudo hacer que el proceso se ejecute en un entorno de *continuous integration* y se realicen validaciones automáticas de futuros diseños.

Además, hubo mejorías en el proceso de generación de bitstream obteniendo los siguientes resultados:

	Punto de partida	Nuevo diseño
Worst Slack	-2.2ns	0.2ns
Duracion del proceso	1 hora	25 minutos

En cuanto a la metodología de trabajo, se concluye que se pudo haber evidenciado antes la localización del problema sin realizar pruebas en hardware ya que los reportes de timing del proceso de PnR indicaban un problema en los caminos de señal relacionados con el core AXItoAHB. Igualmente, las pruebas y herramientas realizadas para identificar el problema sirvieron posteriormente para corroborar el correcto funcionamiento de la nueva versión.

La conclusión u observación más importante que se puede sacar de este trabajo es que es imprescindible verificar en los cores desarrollados por terceros y no confiar ciegamente en los proveedores. Los cores que proporcionan muchas veces tienen fallas, no fueron correctamente validados o no cumplen las especificaciones.

REFERENCES

[1] ARM. (2013) AMBA AXI and ACE Protocol Specification. [Online]. Available: http://www.arm.com/

[2] ——. (2006) AMBA 3 AHB-Lite Protocol. [Online]. Available: http://www.arm.com/

[3] Altera, "Understanding Metastability in FPGAs," Altera, Tech. Rep., 1999.

[4] Microsemi, "HB0397 - CoreAXItoAHBL," Microsemi, Application Note, 2016.

[5] C. E. Cummings, "Clock Domain Crossing (CDC) Design & Verification Techniques Using SystemVerilog," in *SNUG Boston*, 2008.

An FPGA implementation of a Quantum Analog to Digital Converter

Marcos E. Bierzychudek, Rodrigo Melo, Bruno Valinoti, Ricardo Iuzzolino,
Allan Belcher, Jane Ireland, Jonathan Williams and Stephen Protheroe

***Abstract*—This paper presents the development of a Field Programmable Gate Array (FPGA) hardware description applied to a novel Analog-to-Digital converted based on a Josephson Arbitrary Waveform Synthesizer. The aim of this system is to reduce uncertainties and obtain direct traceability for voltage waveform measurements at input frequencies up to 1 MHz. The FPGA controls a high frequency ADC and an optical transceiver which is used to drive a Josephson junction array (JJA) at 10 Gbps. It also filters the ADC readings and transfers the measured data to a PC via 1 Gbps-Ethernet.**

***Index Terms*—Analog-digital conversion, FPGA, Josephson effect, Signal sampling, Sigma delta, Voltage measurement.**

I. Introduction

High-accuracy analog-to-digital (ADC) and digital-to-analog (DAC) converters are used nowadays in many applications, for example in audio players, electrical energy measurements and in medical devices. These components must be characterized in terms of effective resolution, noise-free bits, distortion and other parameters. The electrical metrology community is concerned with the traceability and uncertainty of these measurements in order to ensure their quality and allow the development of improved instrumentation and devices [1].

The EMPIR joint research project 15SIB04 QuADC [2] aims to develop a quantum-accurate ADC based on Josephson junction arrays (JJAs) in order to reduce the uncertainty of voltage waveform measurements up to input frequencies of 1 MHz. The target uncertainty levels require 17 effective bits for high frequency signals and 27 bits for low frequency input signals, up to 1 kHz.

The core of this novel ADC is a Josephson Arbitrary Waveform Synthesizer (JAWS) [3], based on a JJA consisting of SNS (superconductor - normal metal - superconductor) junctions. When a high frequency current pulse is applied to the JJA it generates a quantized voltage-time integral pulse. Therefore, a quantum accurate voltage is obtained using a pulse pattern, i.e. pulse width modulation (PWM), and averaging these quantized voltage pulses over time. In this way, the system generates low distortion signals using high frequency binary pulses.

A simplified block diagram of the QuADC is presented in Fig. 1. The signal to be measured and a feedback signal are fed to an amplifier where they are subtracted. The amplifier output is filtered with a low-pass filter to push the high frequency noise far away from the signal bandwidth. Then, the filter output is measured with a low resolution quantizer, implemented with a commercial ADC. A decimator filter processes the quantizer readings in order to increase the effective resolution. The feedback signal is generated by the JAWS, which is driven in real-time with a Field Programmable Gate Array (FPGA) that produces a PWM pulse pattern proportionally to the quantizer reading. The PWM codes are stored in a look-up table (LUT) which is addressed by the quantizer readings. The indexed data is loaded into the input register of an optical interface to produce an optical pulse pattern. These pulses are transmitted by an optical fibre to a photodiode used to drive the JJA which is located in a cryostat at 4.2 K. An FPGA is a good solution to perform all this task because it is flexible, it has low latency and can be synchronized with an external signal.

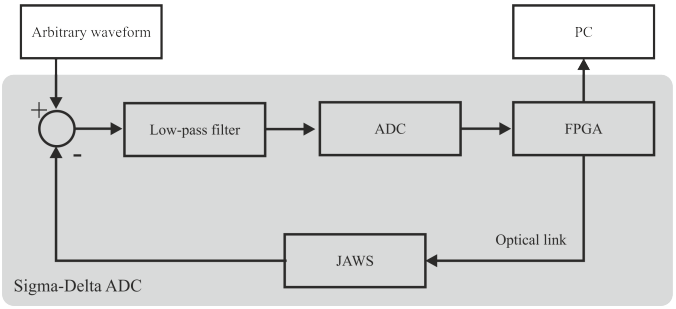


Fig. 1. Simplified block diagram of the QuADC, the shadowed box indicates the sigma-delta ADC. The voltage difference between the generator under test and the JAWS is fed to a low-pass filter. Its output is measured with the ADC and the FPGA drives the JAWS in order to cancel the ADC reading. The measured values are processed in the FPGA and transferred to the PC.

This topology produces a sigma-delta ADC, taking advantage of a low-pass filter to push the noise out of the signal bandwidth and of a decimator filter to eliminate the high frequency noise. In this way, a fast oversampling with a resolution of a single or small number of bits is used with digital filtering to exchange resolution for bandwidth [4]. The low-pass filter was designed as a 4^{th} order continuous-time sigma-delta modulator with a topology of cascade of integrators with feedforward summation (CIFF structure) [5].

II. FPGA design

Fig. 2 presents the system block diagram, including the signal paths and system clocks. The system is controlled by a Xilinx Zynq-7000 FPGA (model XC7Z045 FFG900-2) implemented in the evaluation board ZYNQ ZC706. This device reads the ADC and generates the pulse pattern. These operations are synchronized using three clocks linked via a

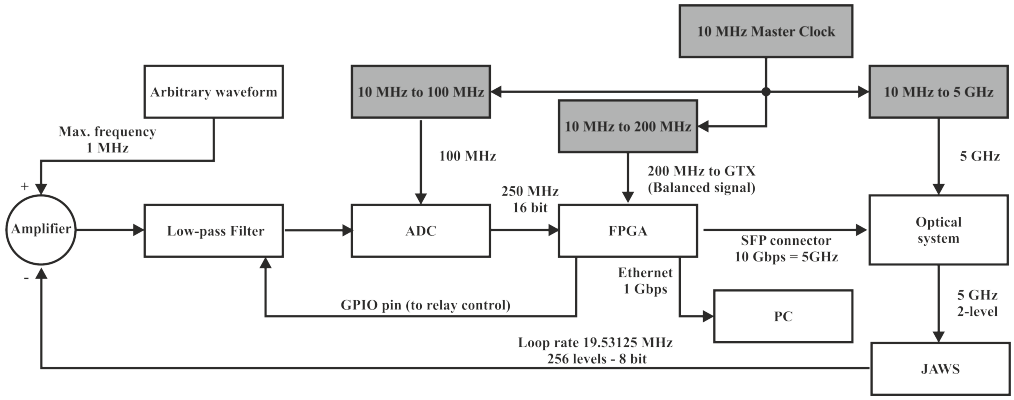


Fig. 2. Block diagram of the QuADC indicating related clocks. The ADC, FPGA and the optical system are synchronized with a 10 MHz master clock and using some clock generators, which are identified in grey.

10 MHz timing link. The FPGA filters the ADC readings and communicates with PC to periodically transfer the processed data. The communication with the PC is performed by 1 Gbps Ethernet. In addition, the FPGA controls some relays included in the filter to reset its capacitors, to zero the voltage input and to invert the voltage inputs. The hardware description is written in VHDL93 and the system is implemented using Vivado 2017.4. Fig. 3 shows the main functional block of the FPGA hardware description.

The quantizer is implemented with the commercial board FMC164 from 4DSP (now Abaco Systems). The ADC board is connected to the FPGA through the FMC VITA 57.1 connector. The IP core FMC16x [6], mainly based on IODELAY and SERDES primitives, was developed to deal with the ADC board.

The Processor System (PS) of the Zynq is used to control the GigaBit Ethernet and 4 million ADC samples can be sent to the PC. The decimator filter was designed as a cascade of two finite impulse response polyphase filters. The first filter reduces the sampling frequency by 2 and the second by 5, giving an equivalent sampling frequency ten times lower. The system has two main operating modes. One is the stand-by and configuration state, where the system can be configured and waits until the measurement start. The other is the measurement mode, where the feedback loop is closed and the FPGA performs the tasks described above.

The JAWS update time, defined by the time difference between two consecutive outputs of the LUT, determines the closed-loop rate and period. In that time interval the FPGA must read the ADC, index the LUT and output the appropriate pulse code to drive the JJA. Therefore, the total delay between acquisition and feedback have to be close to the loop period, its maximum value is 200 ns corresponding to a loop rate of 5 MHz.

A. A Josephson digital-to-analog converter

The Josephson junction array is used by the FPGA as an ideal DAC. In the first prototype, the loop rate is between 5 MHz and 20 MHz, and the JAWS will be driven at 10 Gbps. Therefore, to use the JAWS as a DAC, an upsampling from MHz to GHz is performed by the PWM method, which allows interpolation, increasing the effective resolution. For example, if a loop rate of the order of 20 MHz is required, the pulse pattern for one row of the LUT must have 500 high frequency pulses, (since 10 Gbps$/$20 MHz). This value is replaced with $512 = 2^8$ to use a binary size and the loop rate is adjusted to be 19.53125 MHz. The JAWS system requires return-to-zero (RZ) current pulses in order to generate quantized voltage pulses. Therefore, in between every value in the pulse pattern a data bit equal to zero must be added. The LUT stores 256 configurable values and the RZ pulses are added in software by the FPGA. The mean value of each row gives the corresponding DAC level and the resolution can be calculated as the full scale divided by 256. Thus, an equivalent 8-bit DAC is obtained, that goes from zero up to full scale minus the resolution. When a given ADC reading indexes the table, the corresponding row will be transmitted. After sending the last value of that row, a new ADC reading indexes the table. Therefore the FPGA will send a constant flow of pulses and the output of the implemented DAC does not return to zero.

A sample from the ADC is recorded at a rate equal to the closed loop frequency and the eight most significant bits (MSB) are used as the address of the 256 bit x 256 rows ROM, which implements the LUT. Then, the output of the LUT is RZ coded and sent at 10 Gbps through a GTX transceiver. The output of the GTX is connected to a Small Form-factor Pluggable (SFP+) tranceiver. The pulse sequence is transmitted as raw data with the lowest jitter available, and without pauses or flow control bits. These last ones are not required since the JJA will convert each pulse into a quantized output guaranteeing zero bit lost or data corruption. The GTX transceiver requires a balanced clock input of 200 MHz via specific SMA connectors to generate the 10 Gbps.

B. Clocks and Timing

The system is synchronized with a master clock of 10 MHz from which three other clocks are derived for use as a reference clock for the ADC, the GTX transceiver and the

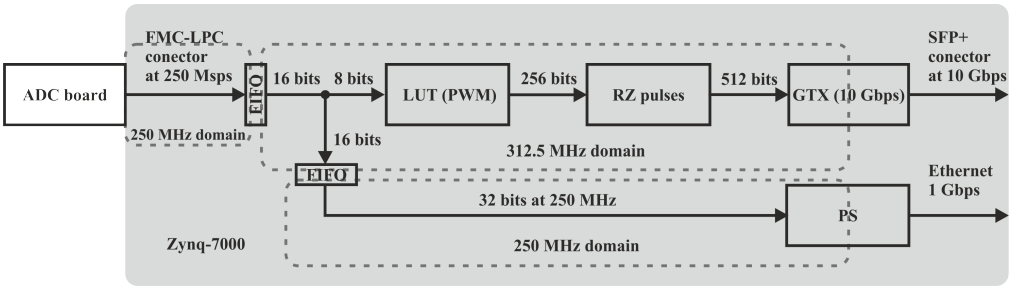


Fig. 3. Block diagram of the FPGA indicating the clock domains (dashed boxes). In the 312.5 MHz domain, the FPGA indexes the LUT, convert to RZ pulses and feeds the GTX transceiver. The processor system is used to communicate with a PC. All the domains are connected with FIFOs.

optical system [7], see Fig. 2. Inside the FPGA, the three subsystems, the FMC16X core, the PS and the SFP, are in unrelated clock domains. As a Clock Domain Crossing (CDC) technique, small FIFOs were used (IN_FIFO primitives). This organization is presented in Fig. 3.

The external clock to the FMC164 board synchronizes the sampling clock and the communication with the FPGA, this clock defines the corresponding domain. The FPGA reads all the time the ADC at 280 MHz. However, to index the LUT the most recent value is used.

The GTX is configured to work at 10 Gbps with 32 bit data packages. The clock of the GTX subsystem is therefore $10\ \text{GHz}/32 = 312.5$ MHz. Each row in the LUT has 256 bits which must be doubled to be converted to RZ pulses, so 512 bits are sent to the GTX in 16 packages of 32 bits.

The measured output of the ADC is transferred to the PC at 250 MHz and it is not synchronized with the master clock. This data is outside of the feedback loop and can therefore be processed at an arbitrary rate without altering the system performance.

C. GTX

The Xilinx 7 Series FPGA Transceivers Wizard was used to configure two identical GTX. One of them is used to output the data at 10 Gbps. The other is used for debugging and is configured as a 5 GHz clock, output via SMA connectors. Both GTX were configured for 32 bit inputs at 312.5 MHz, without any custom coding required. Xilinx recommends the use of an external clock source for the GigaBit transceivers. However, for testing purposes, the GTX Transceiver wizard files were modified to provide an internal clock for the GTX from the PS.

III. Validation and Test

For test purposes, extra functionality was added using push buttons and dip switches of the evaluation board ZC706. Three push buttons were used to guarantee a known pattern at the LUT output, fixing its input as bottom, middle and upper values. The dip switches were used to multiply the number of ADC samples by 2, 4 and 8 respectively.

A Python script was developed to control the IP core and to obtain data. The user can specify the desired quantity of data and the ADC clock source. In addition, the data source can be selected to be either the ADC sample or an internal counter, which is useful to test the data transfer. The data received is saved to a file and another Python script is used to produce graphics in order to examine the recorded date.

The pulse pattern output was examined and some results are presented below. In all the cases, the GTX output was transmitted as an electrical signal via an SMA connector and measured with an oscilloscope of 12.5 GHz of bandwidth. In addition, a pulse signal to identify the start of each LUT row was included when the first data package is sent to the GTX transceiver.

A. Pulse pattern

The output pulse pattern was verified in order to obtain the correct generation. Fig. 4 shows an example obtained with a low frequency sinusoidal signal input. It is clear that the pulse density changes between the consecutive PWM patterns, corresponding to different DAC levels. The pulse pattern was verified by counting the number of positive pulses within a loop period.

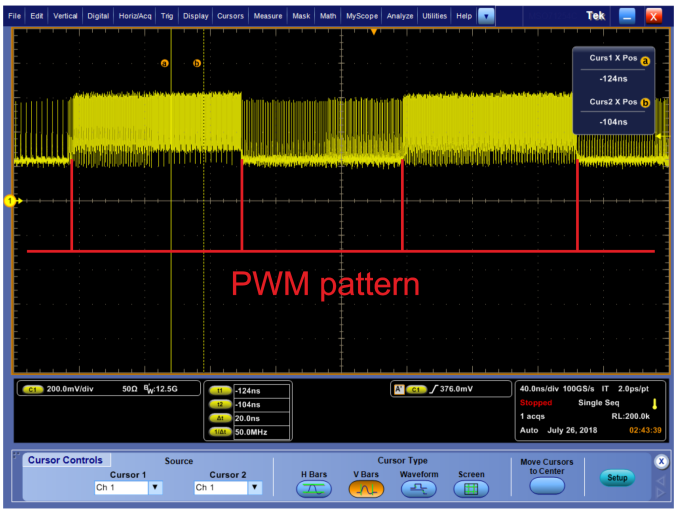


Fig. 4. A pulse pattern corresponding to a low frequency sinusoidal signal, the PWM pattern can be observed.

B. 10 Gbps electrical signal jitter

A pulse train signal was configured in the FPGA and the output signal was measured by the oscilloscope. The measured

jitter was less than 3 ps. Fig. 5 shows an eye diagram of the 10 Gbps output.

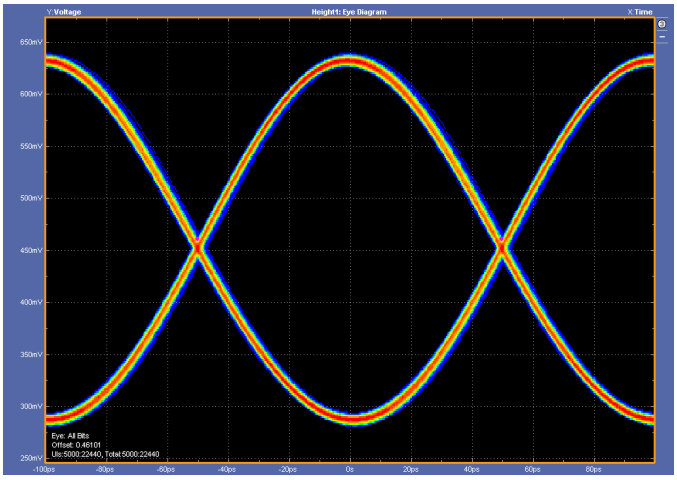


Fig. 5. Eye diagram of the 10 Gbps electrical signal showing a low jitter.

C. GTX delay

To measure the delay time between the LUT row being sent to the GTX transceiver and the output of the first data pulse, the trigger signal described above and the GTX output were examined as shown in Fig. 6. In this test, a full scale pulse pattern was generated and the theoretical mid point of the pattern was found with the cursor 'a' and the trigger signal. The measured mid point of the pattern was found with the cursor 'b' and knowing the theoretical full scale pattern. The time difference between a and b corresponds to the GTX delay time and was found to be 21 ns. This is equivalent to a delay of 6.5 cycles of the 312.5 MHz clock.

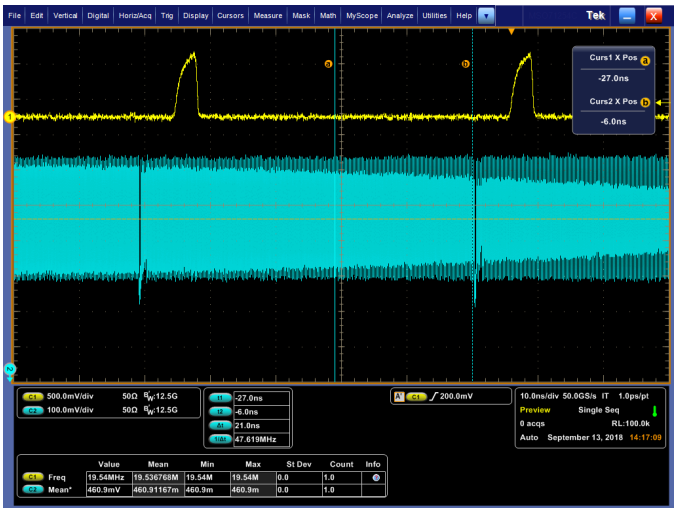


Fig. 6. Measurement of the GTX delay. The upper signal corresponds to the trigger and the lower signal is the GTX output. The pulse pattern was verified counting the number of positive pulses within the PWM pattern.

IV. Results

The whole system includes the QuADC IP core and adds an AXI4-Stream Data FIFO and AXI DMA, both from Xilinx. Also, the PS, the system reset and two AXI interconnect blocks must be considered. As common resources, 3059 LUTs, 4065 FFs and 8 BRAM were occupied. A shared GTXE2_COMMON and two GTXE2_CHANNEL primitives (one for normal data and one as 5GHz clock for test) were used. The ADC clock recovery uses one IDELAYE2 and one ISERDESE2, and additional four ISERDESE2 are consumed for the four data wires. Finally, two IN_FIFO were used for CDC.

V. Conclusion

A quantum analog-to-digital converter is under development with the aim of improving voltage waveform measurement traceability at frequencies up to 1 MHz. The system will be managed with a state-of-the-art FPGA. It reads a low-resolution high speed ADC, generates an optical pulse pattern to drive a Josephson Arbitrary Waveform Synthesizer, filters the ADC data and communicates with the PC. Tests were successfully performed in order to verify correct pulse pattern generation for a given ADC reading. The pulse pattern output was studied and a jitter of 3 ps and a GTX transceiver delay of 21 ns were measured.

Further works will be made in order to improve the design. The most relevant is to implement a new ADC with a higher sampling frequency and smaller delay. The memory size available to record data will be increased and communication with a PC will be implemented by 10 Gbps Ethernet.

Acknowledgment

This work was co-funded by the European Union within the European Metrology Programme for Innovation and Research (EMPIR) joint research project 15SIB04 QuADC. The EMPIR initiative is co-funded by the European Unions Horizon 2020 research and innovation programme and the EMPIR Participating States.

References

[1] A. Belcher, J. Williams, J. Ireland, R. Iuzzolino, M. E. Bierzychudek, R. Dekker, J. Herick, R. Behr, and K. Schaapman, "Towards a Metrology class ADC based on Josephson junction devices," in *IMEKO 2018 - Belfast*, September 2018.

[2] EMPIR Joint Research Project. (2016) Publishable Summary for 15SIB04 QuADC Waveform metrology based on spectrally pure Josephson voltages. [Online]. Available: www.ptb.de/empir/quadc-project.html

[3] J. Kohlmann and R. Behr, *Development of Josephson voltage standards*, Dr. Adir Luiz (Ed.), Ed. InTech, 2011. [Online]. Available: http://www.intechopen.com/books/superconductivity-theory-and-applications/development-of-josephson-voltage-standards

[4] R. Schreier and G. Temes, *Understanding Delta-Sigma Data Converters*. IEEE Press, 2005.

[5] R. Iuzzolino, M. E. Bierzychudek, A. Belcher, R. Dekker, J. Herick, J. Williams, J. Ireland, and K. Schaapmann, "Design and Simulation of a High-order Sigma-Delta Continuous-Time Modulator," in *CPEM 2018 - Paris*, July 2018.

[6] R. A. Melo and B. Valinoti, "Serial QDR LVDS High Speed ADCs on Xilinx Series 7 FPGA," in *Submitted to this conference*, April 2019.

[7] J. Ireland, J. Williams, O. Kieler, R. Behr, E. Houtzager, and R. Hornecker, "An optoelectronic pulse drive for quantum voltage synthesizer," in *CPEM 2018 - Paris*, July 2018.

Packet Core in Mobile Networks: FPGA-based Approach

Santiago E. Nieto, Carlos A. Zerbini, Guillermo G. Riva
Grupo de Investigacion y Transferencia en Electronica Avanzada (GInTEA)
Universidad Tecnologica Nacional, Facultad Regional Cordoba (UTN-FRC)
Cordoba, Argentina
gintea@frc.utn.edu.ar

***Abstract*—Current mobile networking architecture consists of two main components: Radio-Access Network (RAN) and Evolved Packet Core Network (EPC). In particular, current EPC architecture faces performance issues when considering future needs regarding number of devices and traffic types. To solve these problems in current and next-gen (5G) networks, Software-Defined Networking (SDN) and its associated protocol OpenFlow has been applied in previous work. However, these proposals are solely based on software which poses performance constraints. In this paper, we propose to use NetFPGA as hardware accelerator for current and next-gen packet core in mobile networks. For this, we build upon an existing OpenFlow Switch implementation, and evaluate required changes to update this switch and support GPRS Tunneling Protocol (i.e., GTP) on top of it. Considering the widespread of NetFPGA, obtained results demonstrate that this approach has great potential to improve previous proposals for implementing EPCs on top of SDN.**

***Index Terms*—SDN, EPC, OpenFlow, GTP, NetFPGA.**

I. Introduction

LTE, originally defined in 3GPP Rel. 8 (2009) and enhanced in 3GPP Rel. 11 (LTE-A, 2011) and 3GPP Rel. 12 (LTE-B, 2015), is the current mobile platform. In recent years, LTE has suffered a wild growth of traffic flows, mainly due to growing *number and diversity* of users, which makes necessary new proposals to afford this problem. This in turn, involves many different traffic types, such as high-volume data streaming and sporadic low-volume IoT data. 5G, introduced in 3GPP Rel. 15 (2018), addresses many of these challenges and is under intensive discussion.

With recent enhancements in general-purpose hardware capabilities, *Software-Defined* Networking (i.e., *SDN*) and its supporting protocol OpenFlow (OF) has rapidly spread in diverse fields, and in particular for meeting increasing flexibility demands of next-gen (i.e., 5G) mobile networks, where the fundamental concept of *network slicing* arises for isolation and management of highly diverse traffic flows [1]. In this work, we evaluate a hardware-assisted solution for implementing GTP on top of OF switches; achieving line-speed, reconfigurable processing in EPC.

The rest of the paper is organized as follows. In Sec. II we review previous work. In Sec. III, we explain the adopted tools and their application. In Sec. IV we expose the implemented digital architecture and special requirements for GTP, while Sec. V discusses our results. Finally, Sec. VI exposes conclusions about the work done.

This work was done as part of a R&D Project funded by Universidad Tecnologica Nacional. We wish to thank Xilinx Inc. for kind donation of software licenses.

II. Related work

Even if open implementations of the EPC architecture are a relatively recent topic, relevant work exists which gives background to our proposal. Some open implementations of the EPC lead to observing the trade-offs of its implementation and potential issues facing next-generation mobile networks. Notably, we can mention the OpenAirInterface initiative [2].

In [3], authors discuss the re-design of LTE/EPC to fully support OpenFlow, and concentrate on describing how OpenFlow operates for five common GTP procedures. No further details are provided on implementation. In [4], meanwhile, authors adopt a more involved approach with two controllers, one for GTP and other for OpenFlow and also concentrates on EPC procedures without explicit mention of implementation platform and performance issues. In [5], finally, authors observe that current EPCs face scalability issues relating two main aspects, i.e., number of devices and diversification of device types. In particular, they observe that user state is replicated in different entities of the EPC, so intensive signaling traffic is generated during the various events in the network, in particular attachment and migration of devices through the network. To alleviate this problem, authors propose *consolidating* per-user state in one location which they call a *slice*, and *decomposing* EPC functions based on the nature of their accesses. In this way, a user's signaling and data traffic is managed by only one *slice*, and one slice can manage multiple users. In addition, decomposition is achieved by implementing separate *control and data threads* which keep performance isolation required by EPCs. They implement their system on a cluster of top-performance commodity servers, using the NetBricks framework. Extensive evaluation is presented for different traffic traces from [2] and it is compared against four alternative EPC implementations. However, this proposal is exclusively based on software requiring top-performance servers for its goals. In addition, EPC architecture is quite modified which could hinder inter-operation with existing EPCs, and

independent operation of entities at different physical locations is not supported.

III. Proposed approach

A. EPC architecture and GTP protocol

The current EPC implementation is the part of the LTE ecosystem where user packets are processed and finally routed to their final destination. It consists in blocks commonly named as *functions* or *entities*. The core entities are *Mobility Management Entity (MME), Serving Gateway (S-GW) and Packet Gateway (P-GW)*, while additional entities containing system information are *Home Subscriber Database (HSS)* and *Policy Charging Rules Function (PCRF)* (Fig. 1(a)).

MME handles all signaling traffic and works as a central controller of the system. It manages User Equipment (UE) registration, bearer bringup, and handovers. S-GW and P-GW, meanwhile, handle data traffic. The S-GW serves as the local mobility anchor for handovers between eNodeBs, and forwards UE's traffic towards the P-GW. P-GW, finally, is the interface between the EPC and external packet data networks from different service providers. In addition, P-GW provides policy enforcement, filtering, and charging support. UEs can be connected to multiple P-GWs to access multiple service providers, but can be connected to just one S-GW.

EPC traffic is logically divided in *bearers* according to the particular requirements of each traffic type. A bearer is a logical connection between eNodeB and S-GW, and each UE can have one or more associated bearers. In [6], there is a brief guide of how MME, and gateways interact inside architecture.

Main issues of current EPC architecture are, namely [3]:

- The control plane is still tightly coupled with the user (data) plane at S-GW and P-GW.
- A change in the UE state between idle and connected states causes exchange of many signaling messages between network entities.
- Data plane management is performed in a distributed manner which means that a forwarding plane needs to be established for all procedures that require a hierarchical exchange of large number of signaling messages.

For transporting General Packet Radio Service (GPRS) messages within LTE EPC, a group of IP-based communications protocols named GPRS Tunneling Protocols (GTP) is used. In turn, GTP consists of GTP-U, for user data, and GTP-C for signaling traffic. Due to space constraints, we will not elaborate on GTP but will be interested in its main fields and size. Detailed discussion of GTP and its multiple messages can be found in [2], [5], and [7].

B. SDNs and OpenFlow (OF)

SDN is a mature approach which proposes a central controller hosted in some central control place leading switches in a network that can be (or not) in the same place.

The SDN controller contains control functions and uses protocols such as OF to control and configure simple forwarding devices at the data plane, known as *Network Functions (NFs)*. As result, network management is simplified, and

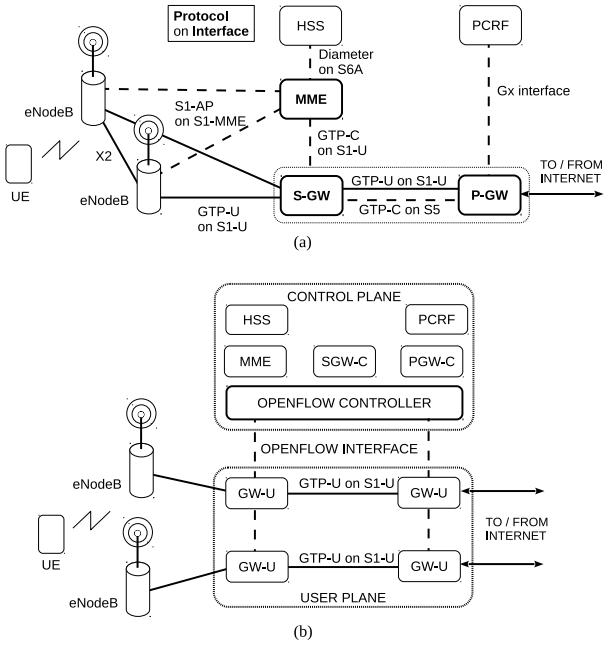


Fig. 1. (a) Current EPC architecture, (b) Proposed approach leveraging OF

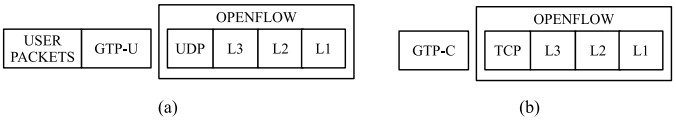


Fig. 2. (a) GTP data plane, and (b) GTP control plane on top of OF

new proposals can be tested faster and simpler. In this way, OF features separation of control and data planes, which enhances programmability. The OF protocol has been adopted by previous work [5] in order to introduce new features to mobile networks keeping backwards compatibility to current deployments. In addition, radio access network of LTE, i.e., E-UTRAN, is not affected by these changes.

C. Distributed, Hardware-assisted architecture

Even if several proposals exist which aim at fitting current EPC to next-gen requirements, they lack two relevant features:

1) On the one hand, they are mostly based on software running on top of general-purpose processors and assisted by virtualization techniques. Even if this fact provides flexibility, they can lack carrier-grade data-plane performance for increasing number and diversity of users. To the best of our knowledge, our proposal is the first one to analyze an FPGA-based platform for this goal.
2) On the other hand, related work assumes that all EPC reside a single software platform. Moreover, [2] integrates S-GW and P-GW in a single entity. This fact can hinder its integration in existing platforms or the strategic location of entities in convenient locations, e.g., where they provide most benefits for the user.

Nowadays, all user traffic is based on IP, and EPC implements its main features such as mobility, QoS, and traffic

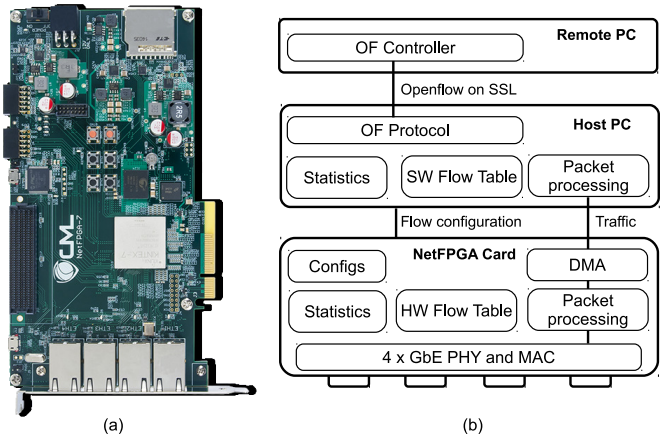


Fig. 3. (a) NetFPGA 1G CML, (b) OF Switch processing path

aggregation through the use of GTP-U tunneling according to rules fixed through GTP-C protocols. That is, user's IP packets enter the eNodeBs through the Radio Access Network (RAN) where they suffer IP-in-IP encapsulation, i.e., they are appended GTP-U headers. These headers, which have semantic meaning only for the EPC, can suffer modifications throughout its path inside it and are removed when they leave it to enter other networks.

Normally, GTP is mounted on classical IP stack. To adopt SDN concepts in mobile networks, we mount GTP on top of OF, i.e., we extend OF protocol to support GTP headers, as illustrated in Fig. 2. In addition, we evaluate hardware requirements in order to support the associated headers on FPGA devices. This approach keeps clear layer separation and achieves high performance for carrier-grade applications. Besides our FPGA-based platform, it could also allow integrating off-the shelf OF-enabled networking devices [3]. In Fig. 1(b) we show this architecture.

IV. Implementation

For our hardware implementation, the NetFPGA board was adopted [9]. NetFPGA is an open source hardware and software platform widely supported by academia and research community. Four versions of this board exist, i.e., two outdated ones including 4x1GbE ports/Virtex II FPGA (NetFPGA 1G) and 4x10GbE ports/Virtex5 FPGA (NetFPGA 10G) respectively, and two new ones including 4x1GbE/Kintex7 (NetFPGA 1G CML) and 4x10GbE/Virtex7 (NetFPGA-SUME) respectively. In particular, for this research, and without loss of generality, we adopt the NetFPGA 1G CML (Fig. 3(a)).

Initially, this board was tested with projects from the official repository. These projects helped getting familiar with the platform by implementing different components on a network, for instance switches and routers, more generically known as *middleboxes* or *network functions*. For our particular goal, we evaluate NetFPGA as a versatile yet efficient OF Switch, which as part of a SDN can support mobile networks like LTE or next-generation 5G [8]. OF switch implementations are available for former 1GbE and 10GbE NetFPGAs. As shown in Fig. 3(b), the OF switch involves three main blocks: a remote OF controller (e.g., NOX) which talks to OF switch through included OF interface; a software-based OF switch which processes low-priority or seldom traffic; and the hardware-based switch which processes high priority traffic at line rate. This project, however, is quite outdated since it only supports OF v1.0 and it has not been ported to current NetFPGA boards.

We base our work on the *NetFPGA 10G OpenFlow Switch* project contributed by Stanford University [9] which is the most recent one (2012), and port it to our board. For this, our work can be divided onto four main stages:

A. Updating OpenFlow pcore to fit the new NetFPGA-1G-CML framework.
B. Re-generating Ternary Content-addressable Memory (TCAM) core for Kintex-7 architecture.
C. Modifying the Flow Control Table.
D. Analyzing OF and GTP packet header requirements, in order to define EPC-specific extensions.

A. Updating OpenFlow pcore

OpenFlow pcore is named "openflow_datapath_v1.0". Inside this core, there are four main blocks as described next and shown in Fig. 4. These modules use 64-bit wide AXI4-Lite (AXI) bus for communication with host, 256-bit AXI Stream (i.e., AXIS) for data paths, and 128-bit AXI TUSER for additional custom signals not propagated outside the pcore.

1) *pkt_preprocessor*: the function of this module is to separate the header and data from an incoming packet and send selected header bits to the flow table, which looks for the respective actions to be done. The rest of the packet is directly sent to *action_processor* block as soon as lookup is ready.
2) *action_processor*: this module receives packets from *pkt_preprocessor* and actions from *flow_ctrl_table*, and uses these data to decide on header values and forwarding port. For this, action tasks await for *lu_req* and *lu_done* signals to be asserted.
3) *host_inf*: this block (not shown for clarity) is the mediator between the OF pcore and the rest of the system. Through this block, the user can update flow tables, see statistics and access the OpenFlow pcore register file.
4) *flow_ctrl_table*: this module receives the header (called *match bits*) of an incoming packet and looks up for the actions to be done in the action block[1]. Specially relevant for our work is knowing that OpenFlow switch has two different flow tables. On the one hand, it has an *Exact Table* where match bits must be equal to the entry fields for matching rules. On the other hand, the *Wildcard Table* is prefix-based, returning *the best of a number of matching entries* with a hierarchical criterion. Both tables return actions, and the *action select* block decides (in

[1]This block ignores packet data. Data from the packet goes directly to action_processor.

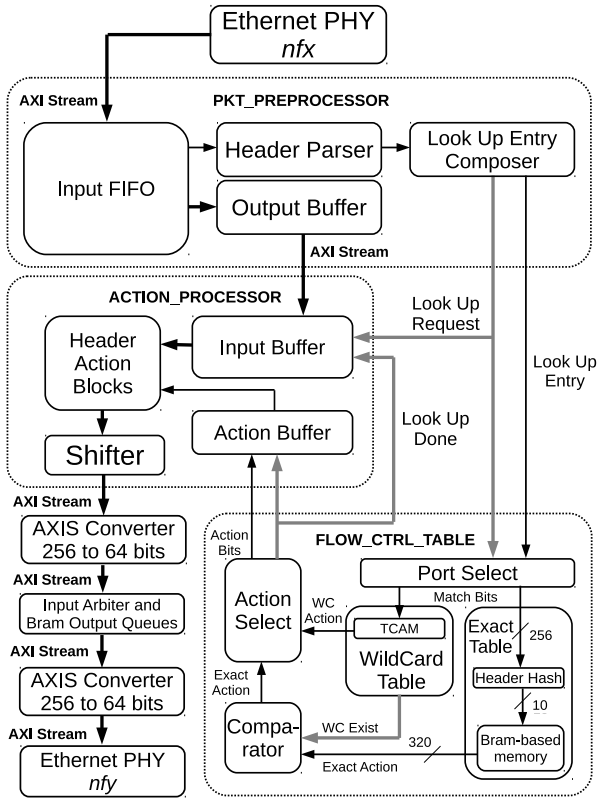


Fig. 4. Processing paths inside NetFPGA OF Switch.

conjunction with the *comparator block*) which action to use.

Using Xilinx XPS as working environment, the first step is adapting buses to integrate the core to the current NetFPGA 1G CML datapath. For this, AXI converters are added since OpenFlow pcore works with 256-bit AXIS internally while the rest of the design works with 64-bit AXIS buses.

OF core includes five *pkt_preprocessor* and five *action_processor* blocks with five ports each. Four ports of each block are mapped to the four physical NIC ports, while the fifth port connects to the PCI_DMA block which transfers packets to/from a SW-based OF switch (see Fig. 3(b)).

Fig. 4 shows the basic processing path, were a packet enters through Ethernet PHY interface *nfx*, and exits through *nfy* if a valid action exists. If not, it is delivered to host PC to take a decision in SW-based OF switch. In general, the input physical port is different than the output one. In Fig. 4, interconnections are represented as follows:

- Thick black lines define main packet path.
- Thin black lines are wires.
- Thick grey lines represent buses.
- Dotted boxes belong to OpenFlow pcore.

B. TCAM Generation

To achieve line-speed prefix lookups, *flow_ctrl_table* uses TCAM memories for storing wildcard flow tables. Since NetFPGA 1G CML includes no native CAM, it uses either

TABLE I
ENTRY/ACTION FIELDS AND FLAG BITS IN OF SWITCH

***Entry* structure**		***Action* structure**		**Action Flags**	
#Bits	**Name**	**#Bits**	**Name**	**Bit**	**Name**
16	transp_dst	16	fwd_mask	0	Output
16	transp_src	**16**	**action_flags**	1	VLAN VID
8	ip_proto	16	vlan_id	2	VLAN PCP
32	ip_dst	8	vlan_pcp	3	POR VLAN
32	ip_src	8	eth_src[6]	4,5	DL src/dst
16	eth_type	8	eth_dst[6]	6,7	New src/dst
8	eth_dst[6]	32	ip_src	8	New TOS
8	eth_src[6]	32	ip_dst	9,10	TO src/dst
8	src_port	8	ip_tos	11	New ECN
8	ip_tos	16	transp_src	12	Push VALN
16	vlan_id	16	transp_dst	13,14	Set/dec TTL
8	pad	8	reserved[18]	15	Reserved

SRL16E or 36-Kb Kintex 7 BRAM blocks for *emulating CAMs* [10], where several parameters need to be well understood and configured. For our case, the basic CAM block is configured as follows:

- Family: Kintex-7.
- Witdh: 32.
- Depth: 32.
- SRL16-Based.
- Standard Ternary mode.
- Simultaneous Read/Write.

The emulation works by mapping match bits to SRL16 address, and storing match results as SRL16 contents. SRL16E (16x1) CAM primitives take a single clock cycle for read (i.e., match) operation, and 16 clock cycles for write operation (one clock cycle per each CAM address). Main reasons for using SRL16-based emulation are: first, *fine granularity*, i.e., less resources are wasted in the implementation; and second, only SRL16E-based emulation allows for ternary modes (i.e., TCAM) according to [10].

C. Modifying Flow Control Table

Every entry of the flow table consists of three main data sections (Table I). The first section involves *match and mask bits* containing information of layers 1 to 4 in the OSI model. The second one holds *action bits* containing information about the field to be modified according to action. The last section stores *stats bits* containing statistic data about entries, used by *host_inf* module. Default sizes of these fields are listed below:

- Match/Mask bits: 256 bits.
- Action bits: 320 bits.
- Stats bits: 64 bits.

As shown in Fig. 4, Flow Control Table is accessed through match bits which are parsed and delivered from the pkt_preprocessor block[2]. The upper part of this structure is the *WildCard Table* (WC) and the lower part is the *Exact Table* (EX), holding 32 and 1024 entries respectively. Exact entries are stored in BRAM blocks, while WC ones are stored in SRL16E-based TCAM blocks. In addition, to reduce

[2]The data bits are sent directly to the action_processor block, which waits for the Flow Table response.

BRAM size and achieve 1-cycle lookup, EX Table uses a hash algorithm for mapping the original 256 match bits down to 10 bits.

In order to access WC table, match bits are segmented and allocated to contiguous emulated TCAM blocks, which in turn assigns them to multiple 4-bit SRL16Es. Moreover, outputs from multiple TCAM blocks must be ANDed since a match is defined as both the lower data bits AND the upper data bits matching the read input on the same address [10]. In this case, Xilinx IDE implements eight 32x32 CAM blocks, consolidating an unique CAM of 32x256. If the same entry matches *both* EX table and WC table, then it uses the action of EX table.

D. Analysis of OF and GTP header support

We discuss required header lengths (i.e., match bits) for our particular implementation. On the one hand, OF header involves 256 bits for its outdated version 1.0 (v1.0), 384 bits for mid-aged v1.1 and more than 1000 bits for recent v1.5. Moderate increase from OFv1.0 to OFv1.1 are due to additional Ingress Port and Metadata fields, while OFv1.5 introduces IPv6 which substantially increases header width. For our tests, we will consider up to OFv1.1. Match bits are set as 256 bits by default in the current NetFPGA OF Switch, i.e., the current system is not able to parse the complete header of OFv1.1, so match bits must be expanded to at least 384 bits. [11] [3].

On the other hand, we consider the needed bits for supporting GTP protocol in its current version 2 (GTPv2). For this, 128 extra bits are needed in the header[3]. Both OFv1.0 and GTPv2 headers are shown in Fig. 5. In brief, header lengths considered in our tests are:

1) 256 bits for supporting OFv1.0 (default implementation)
2) 384 bits for either OFv1.0+GTPv2 or OFv1.1 alone [3]
3) 520 bits for supporting OFv1.1 + GTPv2 (our main case)
4) 1024 bits for (eventually) supporting OFv1.5 + GTPv2

At HDL level, OpenFlow pcore instantiates BRAM memory blocks generated by *Xilinx Block Memory Generator*, with default sizes (depth x width) 1024x384, 1024x256, 1024x64, 32x384 and 32x256. So, as first step, the available widths were expanded to 512 and 1024. With this, 256, 384, 512 and 1024 match bits can be tested. In addition, this implies changing hash algorithm taking care of CRC32 error probability. E.g., for testing 256 match bits, CRC block has 256-bit and 10-bit input and output ports respectively, and the required form factor for BRAM is 2^{10} x match_width = 1024x256.

V. Results

Using the implemented design with the introduced modifications, we evaluate real feasibility of implementing EPC functions on top of OpenFlow using NetFPGA as hardware platform. In particular, NetFPGA 1G CML includes a Xilinx Kintex 7 XC7K325T FPGA which is a mid-range device, and even if the board includes external SRAM and DRAM we use exclusively FPGA memory. We focus on two aspects, namely:

1) Hardware requirements for a typical implementation of OF Switch
2) How these requirements scale when adding fields for supporting GTP on top of OpenFlow

As first case, 256-bit match field is considered and results are listed in Table II. It is notable 268 BRAM blocks, i.e., less than 60% of the available BRAM blocks are necessary in this case for the OF switch. It represents a big advantage with respect to the original NetFPGA-10G implementation (Virtex 5) which uses more than 77% of BRAM blocks for this case. From this first observation we confirm that, as consequence of evolving technology, extended field lengths can be implemented.

As second step, we vary match width from 256 to 384, 512, and 1024 bits, keeping constant number of entries for both EX and WC tables. Results are also listed in Table II. As seen, the most affected resource consumption for increasing match width are Slice Registers and BRAM Blocks. However, it is notable that increasing header width has much lower incremental cost than increasing either EX or WC entries as we demonstrate later. This means that match bits can be extended as much as the designer needs for a particular propose. Anyway, we must note that extending header width beyond 1024 bits without changing the hash module can drastically affect its false positive rate.

Considering requirements and consumed resources, we conclude that 512 is best suited for match width, mainly because we wish to support version 2 of GTP protocol and modern OF v1.1 is intended instead of OF v1.0. This means that 128 bits in OSI model layers and 128 extra bits for GTPv2 are needed in addition to the current NetFPGA OF implementation. Meanwhile, newer versions of OpenFlow protocol would require up to 1024 bits of header, and changing hash module considering the increase in CRC32 errors.

As third step, we consider varying EX and WC tables depth, i.e., number of entries. In Table III, the number of EX entries is varied. The most relevant changes as result of this is BRAM consumption. It is worth noting that RAM memories were

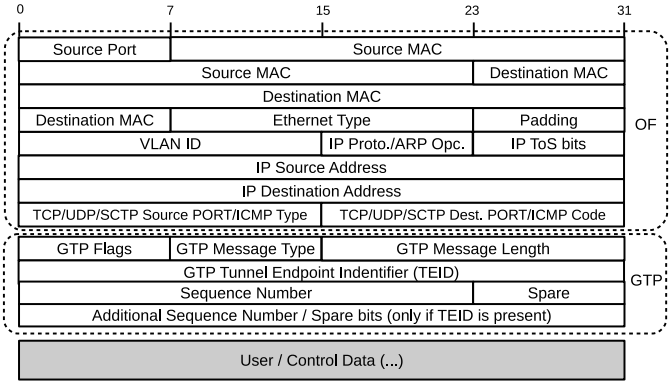


Fig. 5. OpenFlow v1.0 and GTPv2 packet header fields

[3]TEID field appears only if TEID flag (T) is up. If no TEID is present, following fields Sequence Number and Spare, use 64 bits instead of 128.

TABLE II
OVERALL HARDWARE REQUIREMENTS (1024 EX AND 32 WC ENTRIES)

Resource \ Header	256	384	512	1024	Avail.
Registers (Total)	65859	67574	69302	77163	407600
Registers (OpenFlow)	24492	25791	27087	33220	407600
LUTs (Total)	69574	72714	76304	90943	203800
LUTs (OpenFlow)	21491	22031	22517	24740	203800
LUTs (as logic)	63818	64910	66452	72899	203800
LUTs (as RAM)	1340	1340	1340	1340	64000
LUTs (as SRL16E)	4416	6464	8512	16704	64000
BRAM Blocks (Total)	268	271	275	290	445
36x1 BRAM	235	239	242	256	445
18x1 BRAM	65	64	66	68	890

TABLE III
RESOURCES FOR VARYING EXACT ENTRIES (512 MATCH BITS)

Resource \ Ex. Entries	1024	2048	4096
Registers (Total)	69302	69311	69320
Registers (OpenFlow)	27087	27096	27105
LUTs (Total)	76304	76351	76400
LUTs (OF)	22517	22564	22613
LUTs (as logic)	66452	66499	66548
LUTs (as RAM)	1340	1340	1340
LUTs (as SRL16E)	8512	8512	8512
BRAM Blocks (Total)	275	302	355
36x1 BRAM	242	268	321
18x1 BRAM	66	67	67

TABLE IV
RESOURCES FOR VARYING WC ENTRIES (512 MATCH BITS)

Resource \ WC Entries	32	64	128
Registers (Total)	69320	69932	71008
Registers (OpenFlow)	27105	27141	27209
LUTs (Total)	76400	86194	106495
LUTs (OpenFlow)	22613	22775	23092
LUTs (as logic)	66548	68150	72067
LUTs (as RAM)	1340	1340	1340
LUTs (as SRL16E)	8512	16704	33088
BRAM Blocks (Total)	355	355	355
36x1 BRAM	321	321	321
18x1 BRAM	67	67	67

implemented using 8Kx2 primitives. An important observation is that hash output length is incremented in one bit every time exact entries are duplicated (10 bits for the case of 1024 exact entries, 11 bits for 2048 entries, etc).

In Table IV the number of WC entries is varied. Considering results of Table III, exact entries are set as 4096 for this analysis, that is why 355 Bram blocks are used. For increasing WC entries, SRL16s consumption is notably increased; we can estimate that for twice WC entries, SRL consumption is duplicated too. Every time wildcard entries are changed, the method in [10] must be used to generate new CAM blocks, in this particular case 32x64 and 32x128.

Finally, we explore border cases where some resource could be exhausted. For example, bringing exact entries up to 8192 results in 100 additional BRAM blocks needed, reaching BRAM limit for this FPGA (i.e., 445). Similarly, if wildcard entries is set to 256, approximately 67000 SRLs will be required while the FPGA has 64000. As conclusion, the best trade-off for our current purposes is reached using 4096 exact entries and 128 wildcard entries with a header width of 512 bits.

Even if no extensive timing evaluation is discussed in this work, we check that the design can sustain line-speed processing on the NetFPGA-1G-CML platform based on post-place and route data. Considering 512 bits as smallest, worst-case packet length, processing speed of $1 \cdot 10^9 \ / \ 512 \approx 2$ MHz (i.e., Mpps) is needed. Moreover, considering 8-stage lookup pipeline of OF switch [9], worst-case speed scales to $8 \times 2 \cdot 10^6 = 16$ MHz. From post- place and route reports, maximum speed of 45 MHz can be achieved for the whole switch with no timing optimization at all. This confirms its factibility for our case of 1 Gbps, and shows good potential to work at higher speeds or aggregated traffic from multiple ports.

VI. CONCLUSIONS

In this work, an implementation of the GTP protocol on top of OpenFlow is evaluated using the popular, high-performance hardware platform NetFPGA. This implementation has good potential to be applied in real 4G and 5G testbeds to make the EPC architecture more flexible and efficient for future needs.

An OpenFlow core, available for previous versions of NetFPGA, is updated and adapted to implement GTP. Finally, its capacity to support additional header bits and number of rules is evaluated through synthesis.

As future work, a flow table pipeline will be tested where flow sub-tables are implemented for individual fields [11], which provides extra flexibility. In addition, a testbed will be built for testing the switch with real EPC traffic.

REFERENCES

[1] X. Foukas, G. Patounas, A. Elmokashfi and M. K. Marina, "Network Slicing in 5G: Survey and Challenges, " in IEEE Communications Magazine, vol. 55, no. 5, pp. 94-100, May 2017.

[2] N. Nikaein, M. K. Marina, S. Manickam, A. Dawson, R. Knopp, and C. Bonnet, "OpenAirInterface: A Flexible Platform for 5G Research, " SIGCOMM Comput. Commun. rev. 44, no. 5, pp.33-38, Oct. 2014.

[3] V. Nguyen and Y. Kim, "Proposal and evaluation of SDN-based mobile packet core networks, " EURASIP J. Wireless Comm. and Networking 2015 (2015): 172.

[4] A. Mahmoud, A. Abo Naser, M. AbuAmara, T. Sheltami, N. Nasser, "Softwaredefined networking approach for enhanced evolved packet core network, ' Int J Commun Syst. 2018;31:e3379.

[5] Z. Ayyub Qazi, M. Walls, A. Panda, V. Sekar, S. Ratnasamy, and S. Shenker, "A High Performance Packet Core for Next Generation Cellular Networks," In Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17), ACM, New York, NY, USA, pp. 348-361, 2017.

[6] "EPC User's Guide, ' EURECOM, Aug. 2016.

[7] "3GPP TS 29.060 version 12.6.0 Release 12, " ETSI, 2014.

[8] A. Khan and N. Dave, "Enabling Hardware Exploration in Software-Defined Networking: A Flexible, Portable OpenFlow Switch," 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, Seattle, WA, 2013, pp. 145-148.

[9] T. Liu, "Implementing OpenFlow Switch Using FPGA-based Platform, " Master's Thesis, 2014.

[10] K. Locke, "Parameterizable Content-Addressable Memory, " Xilinx XAPP 1151, 2011.

[11] "OpenFlow Switch Specification Version 1.5.1, " Open Networking Foundation (ONF), 2015.

BitSync: A novel Data to Clock Phase Alignment for Microsemi FPGAs

Andrés Miguel Airabella, David Caruso, Andrés Julio Demski
FPGA Engineer
Satellogic S.A.
Buenos Aires, Argentina
Email: {aairabella, david, ademski}@satellogic.com

***Abstract*—In this work a novel method for oversampling and syncing Low-Voltage Differential Signals (LVDS) data between an image sensor and a Microsemi SmartFusion2 Field Programmable Gate Array (FPGA) is presented. The method consists of using DDR Registers at 2x data rate to ensure the oversampling and avoiding having multiple clock sources. An IP-Core for data oversampling, registering, decision and syncing was developed. Experimental results of pictures taken by the system with and without the developed Ip-Core are presented to validate the proposal.**

***Index Terms*—bit sync, clock sync, data sync, high-speed communication**

I. Introduction

In digital synchronous systems, data signals are synchronized using a reference clock. In the rising, falling or both edges, the data are assumed to be stable. In printed circuit boards (PCB) the clock lines and data lines may differ in length and other physical parameters that introduce phase shifts between the data or the clock. The delay may occur between two or more data lines or between data lines and clock.

In Fig. 1 a simplified communication scheme between an Field Programmable Gate Array (FPGA) and a Data Generation Device (DGD) is depicted. In this scheme, the FPGA provides an Output Reference Clock, which DGD uses as reference for generating the data that then input to the FPGA. In the PCB or inside the FPGA, there are other variables that affect the alignment between data and clock, and also between different data lines. They are known as Process, Voltage and Temperature (PVT) [1]. This PVT variation may affect timming relationship of the data link interface (even in transmitter or receiver side). Additionally, the higher data transfer speed the more complicated the synchronization process become.

Usually, some high speed data systems have a feedback clock, that is used internally in the FPGA to sync the data, as shown in Fig. 1. A common approach is to design the PCB to allow Input Feedback Clock having the same physical characteristics than the data it synchronizes. However, depending on the board design or the transmitter, this synchronization clock may not be available (i.e. the FPGA has not enough clock inputs or the PCB has routing problems with clock lines). As in this work, when input feedback clock is not present, another method to synchronize the data must be used.

Some work has been proposed to solve the Data to Clock Phase Alignment. In [2], [3] a method of capturing asynchronous communication using LVDS with $SelectIO^{TM}$ interface Xilinx primitives is described. It has the disadvantage that it uses a primitive that is only present in Xilinx Zynq Devices. Also, this method requires a clock manager for shifting phases in local clocks, that may not be present in other devices, and it isn't present in SmartFusio2 FPGAs.

In [4] is presented a method that consists in sampling the data on four phases of the same clock, then deciding which phase is the most "valid", then re-synchronize the data to the system clock. This method is resource efficient, requiring only two global buffers for clock and one Digital Clock Manager (DCM) in Xilinx Spartan Devices.

In this work a novel method for oversampling and syncing LVDS Data between an image sensor and a Microsemi SmartFusion2 FPGA is presented. The method consists of using DDR Registers at 2x data rate to ensure the oversampling and avoiding having multiple clock sources.

II. Problem Description

In this work, the DGD device depicted in Fig. 1 consists in a image sensor. The sensor is connected to the FPGA using multiple LVDS DDR data signals. The FPGA provides a reference clock to the sensor, but the feedback clock from the sensor is not available in the FPGA. Furthermore, the length of the LVDS signals cannot be ensured to be the same. This situation is depicted in Fig. 2.

In this case, an extra effort must be done at the input stage, in order to ensure the correct syncing of data and internal FPGA clock. In Microsemi FPGA there is not any internal resource that allow syncing external data with internal clock, so an IP-Core had to be developed.

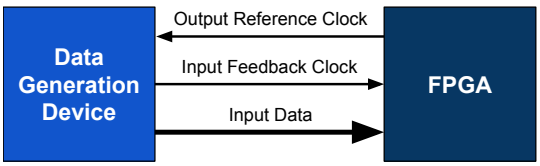


Fig. 1. Simplified Data Communication Scheme

978-1-7281-1363-0/19/$31.00 ©2019 IEEE

A. Timing and Power Consumption

As described in Section I, 4x oversampling of input data is the most common approach when syncing external data. This can be performed by:

- Having a 4x data frequency clock and sampling in the rising edge. This increases the power consumption and complicates the internal FPGA routing.
- Having four 1x data frequency clocks, each one 90° separated from the other, and sampling in the rising edge of them. In this case, power is not an issue, however, the routing becomes complicated and also a Clock Manager (CM) or Phased Locked Loop (PLL) with multiple phase output feature is required.
- Having two 1x data frequency clock, separated 90° and sampling in the rising and falling edge of both clocks. This situation is depicted in Fig. 3. This method requires either 2 DDR Flip-Flops and a PLL capable of providing 90° separated clocks, or 4 Flip-Flops capable of sampling 2 in rising and 2 in falling edges, and a PLL. This method has the disadvantage that uses more resources and more complicated clock routing [2], [3].
- Having a 2x data frequency clock and sampling using a DDR Flip-Flop. This is the method used in this work, and will be discussed henceforth.

B. Design Objectives

The objective of this work is to synchronize the data between an image sensor and a Microsemi SmartFusion2 FPGA. Multiple LVDS lines enter the FPGA at the same frequency as a reference clock, however the clock may not be aligned with the data. The data lines must be synchronized inside the FPGA with the data clock, for further processing. Finally, data coming from the sensor is DDR at **60 Mhz**, that means that a new data is present in both, rising and falling edges of the **60 Mhz** clock and the data rate is **120 Mhz**.

III. Novel Data to Clock Phase Alignment

In this work, a novel method for oversampling the input data is presented. This method consists in using a DDR Flip-Flop at 2x the data frequency. This case is depicted in Fig. 4.

As Fig. 4 shows, data comes from the sensor at **60 Mhz DDR**. In each edge of the **60 Mhz** clock, the data changes.

As the effective data frequency is **120 Mhz**, a **240 Mhz** clock frequency should be used to over-sample the data channels using a DDR Flip-Flop. The bottom waveform of

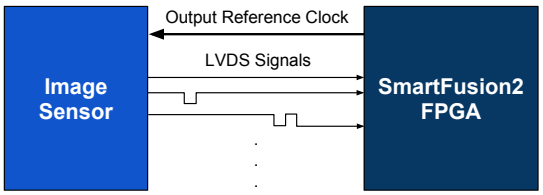


Fig. 2. Data Communication Scheme: Picture sensor feeding a SmartFusion2 FPGA

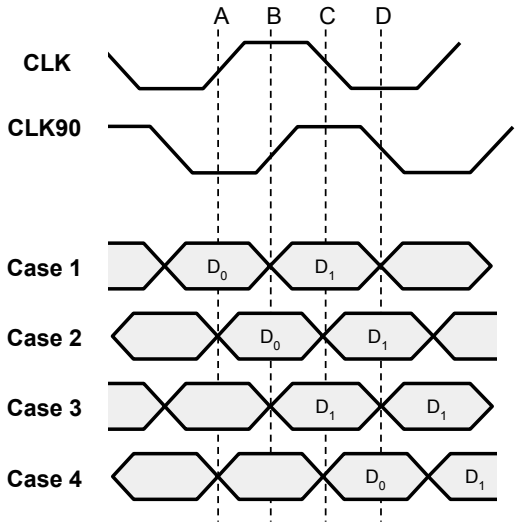


Fig. 3. Data to Clock Phase Alignment, 2 clocks at 90°

Fig. 4 shows a 240 Mhz clock, and the instants *A, B, C* and *D* when data channels are sampled.

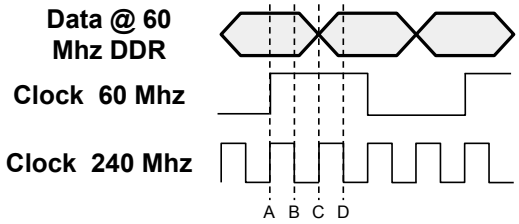


Fig. 4. Waveforms of DDR data and 4x Sampling

In Sub-Section III-A the use of a DDR Flip-Flop for oversampling is presented. Then, a Registering Stage and Edge Detection Stage is described in sub-Section III-B. Finally, in Sub-Section III-C the Decision and Syncing Stage is presented.

A. Using DDR Flip Flops for oversampling

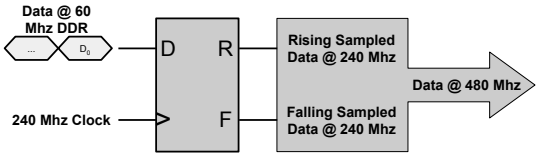


Fig. 5. Input Stage

In order to accomplish the DDR oversampling method, a scheme as shown in Fig. 5 must be implemented. As figure shows, a DDR Flip-Flop [5] receives the DDR data coming form the sensor at **60 Mhz** and over-sample it using a **240 Mhz** clock. In this work, many stages must be instantiated, one for every LVDS line.

At the output of DDR Flip-Flop, two signals are present, *R* for rising sampled data (corresponding to instants *A* and *C* in Fig. 4), and *F* for falling sampled data (corresponding to instants *B* and *D* in Fig. 4). Together, both output channels

produce an equivalent amount of data at 480 Mhz frequency, this is, data at 120 Mhz over-sampled 4x.

B. Registering Stage

In Section III-A the data oversampling using DDR was described. The output data from this DDR Flip-Flop feeds a registering stage, as shown in Fig. 6

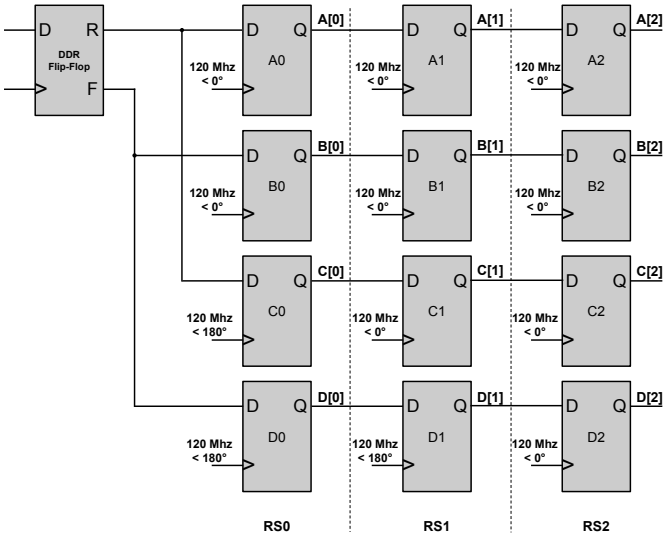


Fig. 6. Registering Stage

After the DDR Flip-Flop, the Registering Stage 0 (RS0), formed by Flip-Flops *A, B, C* and *D* samples the data coming from *R* and *F* signals, but at 120 Mhz, and this is the data rate.

According to Fig. 4, the Flip-Flops in RS0:

- A0 samples instant A.
- B0 samples instant B.
- C0 samples instant C.
- D0 samples instant D.

A0 and *C0* are fed with *R* signal, this is, rising edge sample of data at 480 Mhz, while *B0* and *D0* are fed with *F* signal, this is, falling edge sample of data at 480 Mhz. To acomplish this, *A0* and *C0* use a clock at 120 Mhz with $0°$ offset, and *B0* and *D0* use the same frequency clock but $180°$ offset, as shown in Fig. 7. This $180°$ is possible thanks to the ability of Microsemi Tiles to be confgured as Rising or Falling Edges [6].

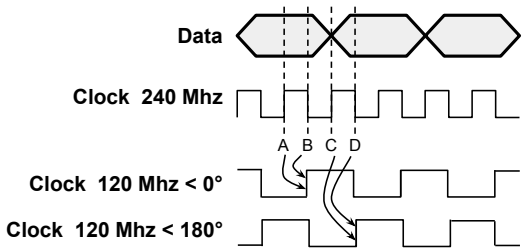


Fig. 7. RS0 Sampling at 120 Mhz

Then, in Registering Stage 1 (RS1) and Registering Stage 2 (RS2), the data is registered again to avoid metastability [4].

The output signals of RS1 and RS2, named *A[1:2]*, *B[1:2]*, *C[1:2]*, *D[1:2]*, will feed the next stage.

C. Decision Stage and Syncing

Signals *A[1:2]*, *B[1:2]*, *C[1:2]*, *D[1:2]* feed eight Edge Detectors, four positive and four negative. The edge detector circuit is shown in Fig. 8, where x = *A, B, C* and *D*. These circuits generate the four positive edge signals, named *AP*, *BP*, *CP*, *DP*, and the four negative edge signals, named *AN*, *BN*, *CN*, *DN*

The edge detection is used to determine between which sampling instants (*A, B, C* and *D* in Fig. 7) occurs a data transition, in order to select the right sampling instant when data is stable.

For instance, using Fig. 7 as example, asume that data changes from "0" to "1" at instant *C*. Negative Edge Detectors will not detect any change, thus their outputs will be "0". Positive Edge Detectors for instants *C* and *D* will not detect any change either, since no change has been produced during these instants, thus their outputs will be "0". Finally, Positive Edge Detectors for instants *A* and *B* will detect an edge, since the data changed at instant *C*, this is, after instants *A* and *B*. For this example, the output of detectors is: *AP = 1*, *BP = 1*, *CP = 0*, *DP = 0*, *AN = 0*, *BN = 0*, *CN = 0*, *DN = 0*.

Using the output of Edge Detector, a logic function must be performed to determine the valid data sample to use. Table I shows the logic conditions that must be asserted in order to select the appropriate sample of data. For the previous example, as shown in this table, when data changes in instant *C*, the useful data (this is, the more stable data) will be present in instant *A*, or exactly $180°$ after the transition. The rest of Logic Conditions and which sample of data to use is also shown in Table I. The full explanation of working principle of this detection process can be found in [4].

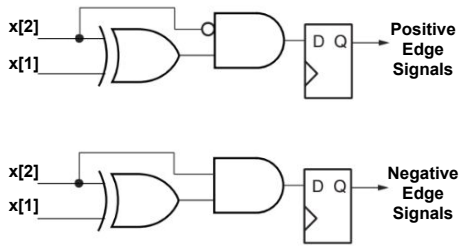


Fig. 8. Positive (top image) and Negative (bottom image) edge detectors.

Table I shows also the signal *Select*. This signal will select one of the four sampled values, which are at the output of the Registering Stage, named as *A[2], B[2], C[2]* and *D[2]*.

Figure 9 shows the output multiplexer. The 4x sampled data pass through four a 4-bit Shift Registers (4 bit SR)s then the multiplexer selects the valid output data. The 4 bit SR sync the data when a transition occurs in the detection process, this is, when the clock and data phase varies over time. The full explanation of working principle of this 4 bit SR can be found in [4].

TABLE I
INTERPRETATION OF EDGES AND VALID DATA CONDITIONS.

Logic Condition	Instant when data changes	Valid data sample	Select Signal
$(AP\&BP\&CP\&DP)$ || $(AN\&BN\&CN\&DN)$	A	C	10
$(AP\&BP\&CP\&\overline{DP})$ || $(AN\&BN\&CN\&\overline{DN})$	B	D	11
$(AP\&BP\&\overline{CP}\&\overline{DP})$ || $(AN\&BN\&\overline{CN}\&\overline{DN})$	C	A	00
$(AP\&BP\&CP\&\overline{DP})$ || $(AN\&BN\&CN\&\overline{DN})$	D	B	01

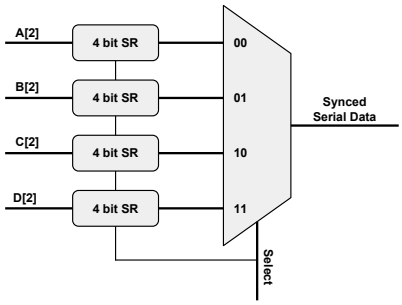


Fig. 9. Output Multiplexer: Selects the correct sampled data.

IV. RESULTS

A. Resource Utilization

A Libero SoC Project was created. The project includes a 33 channels (Sync + 32 Data) inputs and output, driven directly by the image sensor. A local clock of 80 Mhz was used, and a Clock Conditioning Circuit (CCC) was instantiated to generate two clocks of 120Mhz and 240 Mhz.

Table II presents the resource utilization reported by the synthesis tools. As this table shows, a low component count is required. On the other hand, Table III presents the summary of max frequency required. Table III shows the input 80 Mhz clock, *CLK0_PAD*, and the two internal clocks, *FCCC_0GL0*, *FCCC_0GL1*, of 240 and 120 Mhz, respectively.

TABLE II
RESOURCE UTILIZATION

Element	Used	Total	%
4LUT	1651	56520	2.92
DFF	1881	56520	3.33
I/O Register	33	801	4.12
User I/O	101	267	37.83
– Single-ended I/O	35	267	13.11
– Differential I/O Pairs	33	133	24.81
Chip Globals	3	16	18.75
Clock Cond. Circuits	1	6	16.67

TABLE III
MAX. FREQUENCY REQUIREMENTS

Clock Domain	Max Freq. (MHz)	Required Freq. (MHz)
CLK0_PAD	505.051	80.000
FCCC_0GL0	862.069	239.981
FCCC_0GL1	261.712	120.005

B. Experimental Setup

To evaluate the syncing hardware performance, a camera setup (that includes an image sensor and the FPGA) was used and two pictures were taken. The picture shown in Fig. 10 was taken without the syncing hardware present in the FPGA. In this picture, a white line of two pixels wide can be seen, as the corresponding LVDS channel is not correctly synced with the clock inside the FPGA.

On the other hand, Fig. 11 shows a picture taken with the syncing hardware present in the FPGA. As this picture shows, the white lines have disappeared, as the problematic channel is now in sync with its clock.

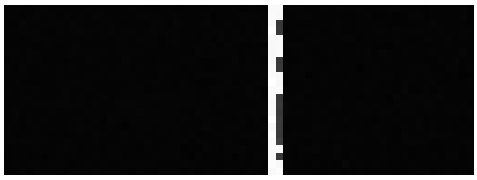


Fig. 10. DarkFrame without the syncing hardware

Fig. 11. DarkFrame with the syncing hardware

V. CONCLUSIONS

The problems associated with syncing data and clock when multiple data lines and clock are present were discussed. A comparison between different clock to data syncing methods was made, and the pros and cons were highlighted.

In this work a novel method for oversampling and syncing LVDS data between an image sensor and a Microsemi SmartFusion2 FPGA was presented. The method consists in using DDR Registers at 2x data rate to ensure the oversampling and avoiding having multiple clock sources. An IP-Core was developed with the following stages: 1) Input stage, were the novel method is included, using a DDR Register. 2) Registering Stage, were the oversampled data is registered at the data frequency, and 3) Decision and Syncing Stage, were transitions in signals are used to determine the best sample of data. The designed IP-Core was tested on a experimental setup. Results of pictures taken by the system with and without the developed Ip-Core are presented to validate the proposal.

REFERENCES

[1] G. W. Starr and X. Zhu, "Clock phase alignment," U.S. Patent 8 650 429, Feb. 11, 2014.

[2] Xilinx, "Xapp523 - lvds 4x asynchronous oversampling using 7 series fpgas and zynq-7000 ap socs," Xilinx, Application Note, 2017.

[3] ——, "Xapp1294 - lightweight and scalable 4x oversampling asynchronous data recovery unit for single-ended or differential inputs," ON Semiconductor, Application Note, 2016.

[4] ——, "Xapp225 - Data to Clock Phase Alignment," Xilinx, Application Note, 2009.

[5] Microsemi, "SmartFusion2 and IGLOO2 Macro Library Guide," Microsemi, User Guide, 2015.

[6] ——, "SmartFusion SoC FPGA Fabric - User Guide," Microsemi, User Guide, 2014.

A new Spiking Neural Network with Extreme Learning for FPGA implementation

1st Iván R. Peralta
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

2nd Nanci Odetti
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

3rd Eduardo Filomena
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

4th Juan I. Rufiner
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

5th Nahuel Ricart
Facultad de Ingeniería
Universidad Nacional de Entre Ríos
Paraná, Argentina

6th Hugo L. Rufiner
Instituto Sinc(i) & Laboratorio de Cibernética
FICH-UNL-CONICET & FI-UNER
Santa Fe, Argentina
lrufiner@sinc.unl.edu.ar

***Abstract*—This paper proposes a parallel fixed point spiking neural network (SNN) implemented in a field programmable gate array (FPGA) with spike response neuron model that makes use of concepts related with extreme learning machines (ELM). For this reason the network is called "Digital Extreme Learning Spiking Neural Network" (DELSNN). The network was designed specifically for its implementation in FPGA. Internal structure and mode of operation are described, where the capability of processing continuous spikes is demonstrated. Matlab numerical calculation software is used for computational model development and training. Then a VHDL model for final implementation in FPGA is generated. For the sake of brevity in this paper only FPGA implementation aspects of DELSNN are presented and discussed. The entire VHDL project was developed using the Xilinx ISE platform, and an Atlys kit board which has a Spartan-6 LX45 as the target FPGA. The device occupancy results for different structures of the network are shown together with the performance achieved in digit recognition task (TIDIGITS).**

***Index Terms*—Spiking Neural Networks, Extreme Learning Machines, FPGA, VHDL.**

I. Introduction

Artificial neural networks are inspired by their biological counterparts and are composed of basic units called neurons. They are interconnected by different weights which represent the intensity of the interaction between each other. Currently, deep artificial neural networks achieve the best performance in virtually any machine learning benchmark ranging from speech recognition to natural language processing to name a few applications [2]. The enormous number of operations that deep artificial neural networks require, represents a great drawback. This limits its execution on platforms with limited processing and energy resources. Currently, their execution is offloaded to remote computer clusters or GPUs. Nevertheless, many applications require fast responses and low-energy consumption as crucial features, for example speech processing in mobile platforms or robotic systems.

Spiking Neural Networks (SNNs) appear as an interesting alternative to simulate large-scale neural networks in real time applications [4], [5]. Actually, neuromorphic engineering [1] allows the emulation of SNNs on hardware in real-time with much higher efficiency in terms of power and speed compared to conventional computing platforms. An example is TrueNorth platform, which is capable of stimulating a million spiking neurons in real-time with only 3 mW of power consumption. The counterpart network on a traditional computing platform was 100, 200 times slower and consumed 100.000 to 300.000 times more energy in each synaptic event [15].

The SNNs were create two decades ago, as the result of searching for a closer analogy to biological neurons. The use of SNN is of special interest in applications that involve a temporal dynamic in the problem to be solved. These have been used on recent applications in many areas of pattern recognition such as visual processing [6], [7], speech recognition [8], [9], [16], and medical diagnosis [10], [11], among others [12], [13]. The aforementioned networks faithfully reproduce the neural biological systems with two effects. On the one hand they try to imitate the transfer of information between neurons through pulses called spikes, in the way it happens in the biological synapses with Action Potentials. Since the neurons communication is done through spikes ('1') or non-spikes ('0'), they are well suited to be implemented on digital hardware. On the other hand, dynamic processing of the signals inside the neurons is done [14] and multiple delayed synaptic terminals can be applied between neurons connections [17]. Thus, they are good candidates to temporal patterns classification.

Aside from the large neuromorphic structures named above, many custom implementations of SNN have been presented in analog and digital hardware [3]. Different architectures have been shown, ranging from wide framework [18], [19] capable to process great custom networks ($> 10,000$ neurons) at many times the real-time speed, to compact systems where small networks are directly implemented onto hardware [20]–[23]. This latter one is used to apply custom solutions to small problems or conform preliminary studies that allow construction of more great SNNs on future.

In an SNN, weighted spikes at the inputs of a neuron are

integrated over time and when that integral exceeds a certain threshold, a corresponding spike is generated at the neurons output. During the training procedure, the strength of the synaptic weights can be changed as result of learning rule application. Actually, a great drawback for use SNNs is their training. For a recent review in learning rules in SNNs, see [5]. One recent paradigm that comes from machine learning community is the Extreme Learning Machine (ELM) [24]. In this structure the connections between the input and hidden layers are randomly weighted, and fixed (they are not altered during training). This strategy is very useful in classification tasks because only the weights of a single layer are trained. The purpose of random weights is to project the signals coming from the first layer in a larger space (by a complex non-linear transformation). The classes that are difficult to separate in the original space are projected to a larger space to allow them to be easily separable in the new representation space.

In this paper we present DELSNN (Digital Extreme Learning Spiking Neural Network) and its compact FPGA implementation. It was designed as an alternative SNN for patterns classification in a supervised manner, using concepts related to ELM. This paper describes the internal structure and mode of operation of the aforementioned network, together with its implementation in FPGA. Due to the require length constraints of this paper, the training algorithm and its applications will not be shown. The reader is referred to [29] to delve into these aspects of the network.

II. DELSNN

The proposed SNN has a *feedforward* architecture with two layers of neurons (**I** and **J**) and an input pattern that can be reduced, for a given instant, to a vector **V** of N components (see Fig. 1). Each neuron of **I** layer connects to each vector elements V_v and each neuron in output layer **J** receives connections from all neurons of layer **I**. Connections between layers are composed of several delay propagation lines weighted by a particular weight for each delay. In one connection between first layer (**I**) and the second layer (**J**), the k-th connection delay between i and j neurons is characterized for a d^k delay and a weight W_{ij}^k (see Fig. 1). The network is designed to perform data classification (encoded in spikes) which are grouped into different classes. In a trained DELSNN, each output neuron represents a possible class and if an example of that class is presented to the network, only the neuron associated with that class must emit spikes. The weights that link the first and second layer of neurons are set randomly, while the weights of the second layer are trained with the examples of the training set. It is worth noting that DELSNN was designed from the beginning with its FPGA implementation in mind, hence explanation of its mode operation is will done in the domain of discrete time.

A. Neuronal Model

Diverse neuron models have been proposed such as the spike response model (SRM) [25], the Izhikevich neuron

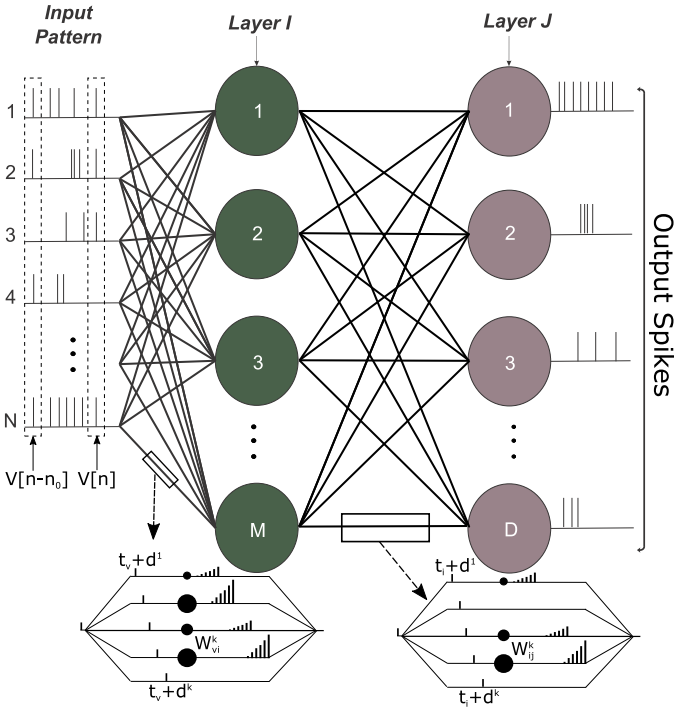


Fig. 1. Internal structure of DELSNN.

model [26], and the leaky integrated-and-fire (LIF) neuron [27]. The neuronal model used for this work is a variant of the impulse response model (SRM). Below, the operation of a neuron i belonging to the input layer is described, but this explanation can be extended to any neuron of the SNN. The state of the neuron i is described by the state variable u_i. This neuron fires if u_i reaches the threshold ϑ. At the instant that threshold is crossed, the discrete firing time $n_i^{(f)}$ is defined, where f indicates the firing number or spike emitted by the neuron i. The set of all firing times of the neuron i is defined by:

$$\mathcal{F}_i = \left\{ n_i^{(f)} \right\} = \{ n / u_i[n] > \vartheta \} . \tag{1}$$

A sequence of spikes emitted by a neuron of the layer **I** can be written as a sequence of discrete Dirac impulses:

$$S_i[n] = \sum_{n_i^{(f)} \in \mathcal{F}_i} \delta[n - n_i^{(f)}], \; with \; \mathcal{F}_i = \left\{ n_i^{(1)}, ..., n_i^{(s)} \right\} . \tag{2}$$

The inputs to the neuron i are given by the spikes coming from each component of the input vector **V**. In turn, since the input vector is updated at each simulation instant, the time evolution of each **V** component can be described as $V_v[n]$, where n is the simulation time and v (with $1 \leq v \leq N$) indicates the v-th vector component.

A spike that belongs to an element V_v of the input vector is distributed in each of the propagation delays d^k of each connection that leaves that element. Once the spikes have been delayed by the propagation delays, they arrive at the neurons as presynaptic pulses that determine a change in the neuronal state. Each presynaptic spike that enters the neuron i, increases

(or decrements) its variable u_i in an amount $W_{vi}^k h[n-d^k]$, where h is the function of impulse response (spike) of the neuron. The increase or decrease of the variable depends on whether the weight is positive or negative, respectively. The state u_i of neuron i at time n is given by the linear superposition of contributions of all propagation delays of all components of input vector: [1]

$$u_i[n] = \sum_{\nu=1}^{N} \sum_{k=1}^{K} \sum_{\tau=0}^{\infty} V_\nu[\tau]\, W_{\nu i}^k\, h[n-\tau-d^k]. \qquad (3)$$

If we consider a neuron of the output layer **J**, the update equation of the state variable u_j of neuron j at time n is given by the linear superposition of contributions of all spikes coming from neurons of layer **I**:

$$u_j[n] = \sum_{i=1}^{M} \sum_{k=1}^{K} \sum_{\tau=0}^{\infty} S_i[n]\, W_{ij}^k\, h[n-\tau-d^k]. \qquad (4)$$

Like the Ec. (2), a sequence of spikes emitted by a neuron of the **J** layer can be written as a sequence of discrete Dirac impulses:

$$S_j[n] = \sum_{n_j^{(f)} \in \mathcal{F}_j} \delta[n-n_j^{(f)}], \qquad \mathcal{F}_j = \left\{ n_j^{(1)}, ..., n_j^{(s)} \right\}, \quad (5)$$

where $n_j^{(f)}$ is the instant of firing number f of neuron $j \in \mathbf{J}$.

Taking into account Ec. (3) and (4), one can interpret the internal functioning of a neuron, that is, the variation of its state u, as the discrete linear convolution of incoming spike sequences with an impulse response function $h[n]$. This implies that at the moment of simulating the SNN, we can choose a $h[n]$ with arbitrary form, which is a potentiality of SRM model.

III. ARCHITECTURE AND IMPLEMENTATION

A complete system diagram is shown in Fig. 2. The SNN is implemented within the FPGA, whose operation is controlled by the control unit (GENERAL CONTROL UNIT). This unit is in charge of controlling and exchanging the incoming and outgoing SNN spikes with the PC through a UART block (Universal Asynchronous Receiver-Transmitter).

For each simulation step, the control unit places in its output *vector_in* a binary vector with the input spikes to the SNN. Then activates the signal *clk_spk*. The SNN processes the input and places in its output *spikes_out* another binary vector with the responses of each of the output neurons for that simulation step. Then, the SNN activates its output *end_processing_SNN* to indicate to control unit that it has finished processing the entries. The control unit is responsible, through the UART, to send these outputs to the PC and to receive the next inputs for the next simulation step.

A *pushbutton* is used to generate a reset signal for the entire system. The *leds* of the board are also used to report possible problems or errors in data transmission.

[1]The internal sum with the upper end ∞ indicates that the sum is done until the end of the whole input pattern.

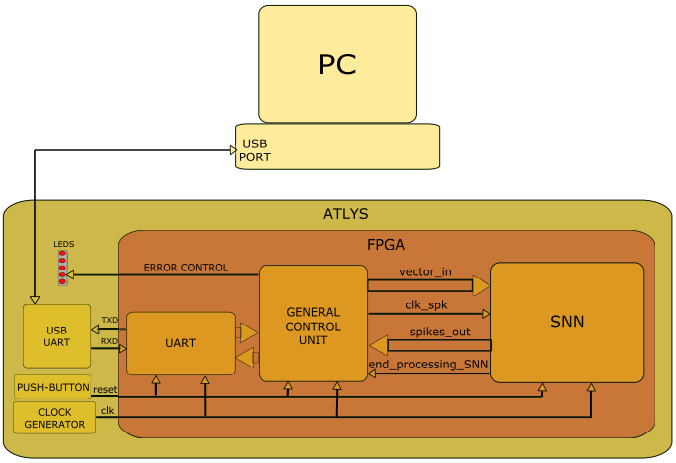


Fig. 2. Block diagram of the complete system implemented in the FPGA, *Atlys* kit board and communication with the PC.

The main objective of this work is to achieve a functioning of the SNN in the FPGA but without emphasizing in the efficiency of operation or the optimal use of FPGA resources. So that some simplifications are made in order to reduce the design complexity: a) The main simplification is that one processing element (PE) is used for each neuron (see [29]). This implies that each PE will only control a single neuron, which significantly reduces the number of neurons that can be implemented in the FPGA. b) The internal registers of each neuron are represented by means of 32-bit integers, which allows the direct use of the data type *integer* of the VHDL language. This simplifies the neuron design, but does not make optimal use of the device resources.

A. Neural Design

The SNN implementation is carried out following a Serial Processing Parallel Arithmetic (SPPA) architecture. That is, an arithmetic is used at the level of bit registers and a serial processing of each connection that enters a neuron. Given that we are in an experimental stage in which we wish to evaluate the SNN performance in response to different responses to the impulse $h[n]$, the architecture designed must be capable of representing any model of neuronal response (SRM).

The SRM model is not a simple model to implement, so there are not many works that have addressed this type of neural model in compact design implementations [20], [23], [28]. The main drawback of this model is that to calculate the membrane potential, the impulse response function must be multiplied by the interconnection weight of each synapse that has an active spike. In addition, in order to simulate the temporal evolution of each connection, the postsynaptic potentials (PSP) of each synapse must be treated in an independent registry, which in a SNN with many connections per neuron, would demand many device resources. Another important aspect is to determine if the neural model supports the application of a spike in its input while the evolution of a previous PSP is occurring. In an extreme case the

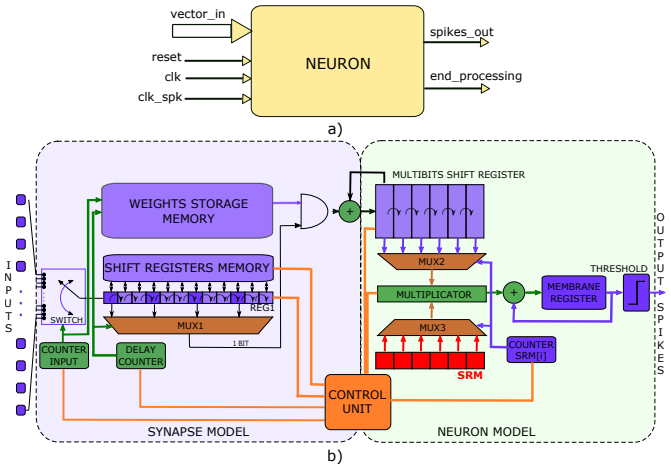


Fig. 3. Neuronal model implemented in the FPGA. (a) Incoming and outgoing signals from a neuron. (b) Simplified schematic diagram of the synapse and neuron model. Thick lines represent multi-bit connections, while thin lines represent one-bit connections.

neuron must support the application of multiple input spikes applied consecutively over time. This is important, since if this capacity is not available, the type of possible coding for the entry patterns to the SNN is limited.

The design proposed in this work notably reduces the complexity of implementation through the use of accumulators and shift registers to represent the evolution of the membrane potential. In addition, it has the ability to apply consecutive spikes over time. This capacity has not been evidenced in any of the models of SRM neurons implemented in FPGA described so far. The Fig. 3 describes the input and output signals of the neuronal model and its internal functioning.

Each neuron receives: the vector *vector_in*, with the input spikes corresponding to a simulation step; the signal *clk_spk*, which defines the simulation period; the signal *clk*, that determines the internal working speed of the neuron and has a much shorter period than *clk_spk* (typical ratio of 1: 10000), and the *reset* signal to manually restart the system. On the other hand, the neuron has two single-bit outputs. The output *spikes_out* describes the outgoing spikes of the neuron and connects itself, depending on the neuronal layer to which it belongs, to the inputs of its postsynaptic neurons or to the vector *spikes_out* that comes out of the SNN described in Fig. 2. It also has as output the signal *end_processing*, which indicates that the neuron has finished processing its inputs. [2] Once all neurons of SNN have finished processing, the signal *end_processing_SNN*, described above, is activated, indicating the end of the processing of the complete network for a particular input vector.

Pressing the *reset* button starts the system. Initially the control unit puts all registers, counters and values of the SHIFT REGISTERS MEMORY to zero, to allow a simulation with null initial conditions. Then, the control unit waits for the activation of the input signal *clk_spk*, which is issued by the GENERAL CONTROL UNIT. This signal tells the SNN to begin processing each of the connections with the presynaptic neurons (in the case that it is a neuron of the last layer) or the input vector (for the neurons of the first layer) as appropriate. Then the first bit of the shift register REG1 is loaded with the logical value present in the first input. The selection of the entry to be processed is done through the COUNTER INPUT. The REG1 determines the propagation delay of a particular input of the *vector_in*. Once the REG1 register has been modified, it is stored in the SHIFT REGISTERS MEMORY for its later reuse in the processing of the next pattern. To map the weights, some outputs of REG1, with specific delays, are connected to a multiplexer MUX1. Then, the output of MUX1 is combined by an AND function with the weight corresponding to that delay. In such a way that if there is a '1' in the selected delay, the weight for that delay will accumulate in the first sub-register of the MULTIBITS SHIFT REGISTER. The weights of the connections are stored in the WEIGHTS STORAGE MEMORY and the address of that memory is determined by the DELAY COUNTER and the COUNTER INPUT. The process described above is repeated until all the inputs of the neuron are scanned.

Once all the inputs have been processed, the membrane potential stored in the MEMBRANE REGISTER is calculated. For this moment, in the first sub-register of the MULTIBITS SHIFT REGISTER, a weight value is stored, which is the sum of all the weights of the active delays associated with each input of the neuron. The impulse response $h[n]$ is stored in the SRM register. Therefore, the first value of the impulse response (SRM[1]) multiplied by said total weight is added to the membrane potential. Then there is a shift to the right of the values of each sub-register of the MULTIBITS SHIFT REGISTER. This allows that in the processing of the next input vector to the neuron, the initial accumulation of weights is not lost and the complete evolution of the PSP associated with that first input vector can be described. Therefore, multiplication is done for all sub-registers of MULTIBITS SHIFT REGISTER. That is, in each processing of an input vector, the multiplication of the i-th sub-register of the MULTIBITS SHIFT REGISTER with the i-th value of the impulse response (SRM [i]) is added to the membrane potential. The MULTIBITS SHIFT REGISTER is what allows the neuron to process contiguous spikes at the entrance of the neuron. Finally, if the MEMBRANE REGISTER exceeds a pre-set threshold, the *spikes_out* output of the neuron is activated.

The CONTROL UNIT, internal to the neuron, was carried out by means of a state machine implemented through a process in VHDL. Its operation is timed by the fast clock signal *clk* and is responsible for both the control of the synaptic model and the neuronal model.

Both the MULTIBITS SHIFT REGISTER and the SRM are composed of 32-bit sub-registers (integers). The SRM register is identical for all neurons and is defined in a configuration file (*configuration.vhd*) together with other design parameters.

The innovation of the design proposed in this paper is that

[2]This output must be differentiated from the output *end_processing_SNN*, which indicates when the complete SNN has finished processing the entries.

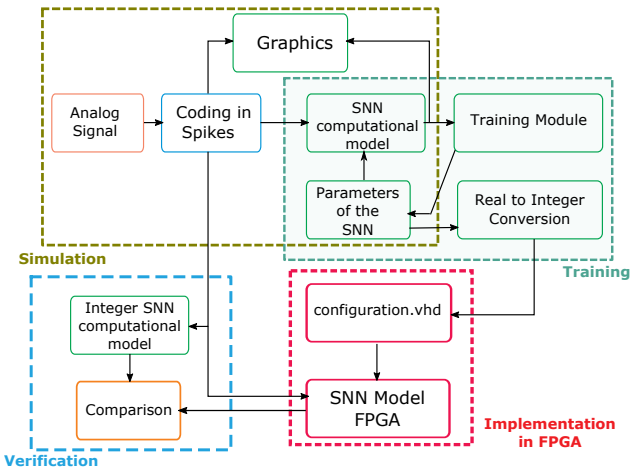


Fig. 4. Conceptual diagram of the different stages of work.

it does not need independent registers to store, separately, the PSP of each entry as it happens in [23]. Instead, an integral treatment of the inputs is carried out, being able to keep all information of the membrane potential in the MULTIBITS SHIFT REGISTER and the MEMBRANE REGISTER. As evidenced, the proposed model uses a serial processing of the neuron inputs, which reduces the complexity of the hardware significantly. However, all neurons are processed in parallel, which allows obtaining a good relationship between the performance and the occupation of the device.

B. SNN module

A block of higher hierarchy than the neurons (called SNN) is defined, which groups all of them into a single structure. To generate said block in VHDL, the neurons are replicated by means of a loop (*for ... generate*) within which the connections between each neuron are defined. The replication of each neuron is done through the use of generic parameters, which are stored in the file *configuration.vhd*. In turn, in this block the signal *end_processing_SNN* is generated by applying an AND logical function of all signals *end_processing* belonging to network neurons.

The storage of the weights of each connection is done in a BROM block, one for each neuron.

IV. Performance Evaluation

The Fig. 4 shows a conceptual scheme of the different stages to evaluate the correct functioning of the SNN. Firstly, a computational model of the SNN was implemented and its operation was evaluated with the excitation of different spike patterns. The computational model was made in MATLAB and uses a floating type arithmetic (real numbers). This model describes the operation of the network, emulating the operation of the SNN implemented in VHDL. Then the weight values are determined. The training of the network is done iteratively in MATLAB (see [29]).

Once the training stage is finished, given that the SNN model in VHDL was programmed with integer arithmetic, it is necessary to convert the values of the weights from real numbers to integers. In Ec. (6) the conversion of the real type weights to the integer type is described.

$$weight_{int} = round\left(\frac{(N_c - 1) * weight_{float}}{MAX - MIN}\right), \quad (6)$$

where N_c is the amount of levels used in the quantization. The values MAX and MIN are the respective maximum and minimum for the whole set weights of the SNN once the training is finished. The $round(x)$ function rounds the value of x to the nearest integer. The Ec. (6) not only modifies the decimal part of the weights, but also modifies the scale of them, so it is necessary to apply the same transformation to the thresholds of fire.

The representation of signed numbers that is used in this work is complement to 2 (one bit to represent the sign and B bits for the magnitude). The number of magnitude bits and the number of levels (N_c) are related by the Ec. (7),

$$B = \log_2(N_c). \quad (7)$$

Once the conversion of the weights is done, we proceed to create the file *configuration.vhd*, which is a text file that groups all configuration parameters of the SNN. The .vhd extension allows it to be integrated into the Xilinx Project Navigator project without any previous steps. In this way, the file *configuration.vhd* becomes the nexus between the computational design in MATLAB of the SNN and the design in the Project Navigator of Xilinx.

Once the design of the SNN is implemented in the FPGA, it is necessary to verify its correct operation. To carry out this verification, a MATLAB *script* was implemented that stimulates with a single pattern the two models of SNN: the computational quantized and the electronic implanted in the FPGA. After exciting both networks, the *script* picks up the output spike sequences of each model and compares them. It is determined that the operation is correct if both responses are equal. The spike responses to same stimulus of both the computational model with its quantized weights (MATLAB) and the electronic model (FPGA) were identical in all experiments, corroborating the correct functioning of DELSNN.

V. Results

The design in VHDL is completely synthesizable on the FPGA, as an example, the Spartan 6 XC6SLX45 device is used with 8-bits weights representation. For comparative purposes Table I shows the resources used in different DELSNN structures. Due to 8-bit quantization, most of the BRAMs were used by the SHIFT REGISTERS MEMORY, which are used for the models of synapses between neurons. For the 32x4x16 and 1920 synapses case, only 30 of the 116 available BRAMs were used. The large number of slices used is mainly due to the fact that all the internal registers of the design are of an integer type (32 bits). It should be noted that this excessive bit resolution is not necessary for most of the internal registers of neurons. All the cases in the Table I were tested with a clock

TABLE I
COMPARISON OF LOGIC RESOURCES UTILIZATION IN DIFFERENT NETWORKS STRUCTURES WITH 8-BITS WEIGHTS REPRESENTATION FOR K=3 (TOP OF EACH ROW) AND K=10 (BOTTOM OF EACH ROW) DELAYS / WEIGHTS PER CONNECTION RESPECTIVELY.

DELSNN Structure	Synapses	Slice Registers	Slice LUTs	Block RAM
2x2x2	24	1599	3152	4
	80	2271	3641	6
9x9x4	351	5112	10028	13
	1170	7296	11673	20
32x4x16	576	7145	11312	20
	1920	10569	14188	30

frequency of 100 MHz and the worst case limits the maximum possible frequency to 117 MHz. Although this paper is limited to describing the implementation of DELSNN in FPGA, it is worth mentioning that in [29] the performance of DELSNN in the task of speech recognition for the classification of isolated digits is shown with results comparable to other state of the art alternatives (TIDIGITS, $WER(\%) = 12,75$). In most experiments only a 5 bits weights quantization was sufficient to not affect significatively the network recognition.

VI. CONCLUSIONS

This work presents a proposal for the parallel fixed point implementation of a new type of SNN in a FPGA. The internal architecture of SNN implementation was shown and analysed. FPGA implementation was carried out satisfactorily, remarking an innovative neural design able to perform continuous spikes processing and low amount of bits required for weigths representation on internal registers. The results obtained were highly satisfactory, indicating the potential feasibility of the technique for use in practical situations. Since that the optimization of the FPGA resources was not a direct goal of this work, it is also possible to perform an explicit optimization of the VHDL design to improve its neurons storage capacity and processing speed in future works.

REFERENCES

[1] Indiveri, G., Horiuchi, T. K. (2011). Frontiers in neuromorphic engineering. Frontiers in neuroscience, 5, 118.

[2] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. Neural networks, 61, 85-117.

[3] Misra, J., Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress. Neurocomputing, 74(1-3), 239-255.

[4] Ponulak, F., Kasinski, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. Acta neurobiologiae experimentalis, 71(4), 409-433.

[5] Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., Maida, A. S. (2018). Deep Learning in Spiking Neural Networks. arXiv preprint arXiv:1804.08150.

[6] Wysoski, S. G., Benuskova, L., Kasabov, N. (2010). Evolving spiking neural networks for audiovisual information processing. Neural Networks, 23(7), 819-835.

[7] Meftah, B., Lezoray, O., Benyettou, A. (2010). Segmentation and edge detection based on spiking neural network model. Neural Processing Letters, 32(2), 131-146.

[8] Tavanaei, A., Maida, A. (2017, November). Bio-inspired multi-layer spiking neural network extracts discriminative features from speech signals. In International Conference on Neural Information Processing (pp. 899-908). Springer, Cham.

[9] Wade, J. J., McDaid, L. J., Santos, J. A., Sayers, H. M. (2010). SWAT: a spiking neural network training algorithm for classification problems. IEEE Transactions on Neural Networks, 21(11), 1817-1830.

[10] Kasabov, N., Feigin, V., Hou, Z. G., Chen, Y., Liang, L., Krishnamurthi, R., ... Parmar, P. (2014). Evolving spiking neural networks for personalised modelling, classification and prediction of spatio-temporal patterns with a case study on stroke. Neurocomputing, 134, 269-279.

[11] Ghosh-Dastidar, S., Adeli, H. (2009). A new supervised learning algorithm for multiple spiking neural networks with application in epilepsy and seizure detection. Neural networks, 22(10), 1419-1431.

[12] Borisyuk, R., Chik, D., Kazanovich, Y., da Silva Gomes, J. (2013). Spiking neural network model for memorizing sequences with forward and backward recall. BioSystems, 112(3), 214-223.

[13] Kasabov, N., Dhoble, K., Nuntalid, N., Indiveri, G. (2013). Dynamic evolving spiking neural networks for on-line spatio-and spectro-temporal pattern recognition. Neural Networks, 41, 188-201.

[14] Gerstner, W., Kistler, W. M. (2002). Spiking neuron models: Single neurons, populations, plasticity. Cambridge university press.

[15] Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., ... Brezzo, B. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. Science, 345(6197), 668-673.

[16] Mir-Amarante, L., Gmez-Rodrguez, F., Jimnez-Fernandez, A., Jimnez-Moreno, G. (2017). A spiking neural network for real-time Spanish vowel phonemes recognition. Neurocomputing, 226, 249-261.

[17] Natschlger, T., Ruf, B. (1998). Spatial and temporal pattern analysis via spiking neurons. Network: Computation in Neural Systems, 9(3), 319-332.

[18] Schoenauer, T., Mehrtash, N., Jahnke, A., Klar, H. (1999, March). MASPINN: Novel concepts for a neuroaccelerator for spiking neural networks. In Ninth Workshop on Virtual Intelligence/Dynamic Neural Networks (Vol. 3728, pp. 87-97). International Society for Optics and Photonics.

[19] Hellmich, H. H., Geike, M., Griep, P., Mahr, P., Rafanelli, M., Klar, H. (2005, July). Emulation engine for spiking neurons and adaptive synaptic weights. In Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on (Vol. 5, pp. 3261-3266). IEEE.

[20] Iakymchuk, T., Rosado, A., Frances, J. V., Batallre, M. (2012, July). Fast spiking neural network architecture for low-cost FPGA devices. In Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2012 7th International Workshop on (pp. 1-6). IEEE.

[21] Moctezuma, J. C., McGeehan, J. P., Nunez-Yanez, J. L. (2015). Biologically compatible neural networks with reconfigurable hardware. Microprocessors and Microsystems, 39(8), 693-703.

[22] Wan, L., Luo, Y., Song, S., Harkin, J., Liu, J. (2016, June). Efficient neuron architecture for FPGA-based spiking neural networks. In Signals and Systems Conference (ISSC), 2016 27th Irish (pp. 1-6). IEEE.

[23] Rosado-Muoz, A., Fijakowski, A. B., Bataller-Mompen, M., Guerrero-Martnez, J. (2011, January). FPGA implementation of spiking neural networks supported by a software design environment. In Proceedings of 18th IFAC World Congress. Milano, Italy Milano, Italy.

[24] Huang, G. B., Zhu, Q. Y., Siew, C. K. (2006). Extreme learning machine: theory and applications. Neurocomputing, 70(1-3), 489-501.

[25] Jolivet, R., Timothy, J., Gerstner, W. (2003). The spike response model: a framework to predict neuronal spike trains. In Artificial Neural Networks and Neural Information Processing ICANN/ICONIP 2003 (pp. 846-853). Springer, Berlin, Heidelberg.

[26] Izhikevich, E. M. (2003). Simple model of spiking neurons. IEEE Transactions on neural networks, 14(6), 1569-1572.

[27] Delorme, A., Gautrais, J., Van Rullen, R., Thorpe, S. (1999). SpikeNET: A simulator for modeling large networks of integrate and fire neurons. Neurocomputing, 26, 989-996.

[28] Johnston, S., Prasad, G., Maguire, L., McGinnity, M. (2005, September). Comparative investigation into classical and spiking neuron implementations on FPGAs. In International Conference on Artificial Neural Networks (pp. 269-274). Springer, Berlin, Heidelberg.

[29] Peralta Iván R. (2017). Reconocimiento de dígitos mediante Redes Neuronales Pulsantes implementadas en FPGA (Tesis de maestría). Facultad de Ingeniería, Universidad Nacional de Entre Ríos (FIUNER), Paraná, Argentina.

A Review of Multi-Camera Tracking Systems Based on Reconfigurable Devices

Farhana Binte Sufi
Applied Physics and Electronic Engineering
University of Rajshahi
Rajshahi, Bangladesh
fsufi.apee@ru.ac.bd

Julio Daniel Dondo Gazzano
Information Technology and Systems
University of Castilla La Mancha
Ciudad Real, Spain
juliodaniel.dondo@uclm.es

Fernando Rincon Calle
Information Technology and Systems
University of Castilla La Mancha
Ciudad Real Spain
fernando.rincon@uclm.es

Juan Carlos Lopez Lopez
Information Technology and Systems
University of Castilla La Mancha
Ciudad Real Spain
juancarlos.lopez@uclm.es

***Abstract*—Real-time video image processing requires video compression techniques. Efficient Motion Estimation (ME) and Motion Compensation (MC) algorithms and their successful hardware implementation are the key to video compression. Work on developing and implementing efficient ME and MC algorithms for multi-camera systems is ongoing. The low power consumption yet high speed of Heterogeneous Reconfigurable Devices such as Field Programmable Gate Arrays (FPGA) can be suitable for implementation of real-time optic flow computation using multi-camera systems. This paper focuses on a search into the current state-of-the-art for multi-camera motion tracking. It has been found the tracking systems mainly focused on people and vehicle tracking in both indoor and outdoor conditions, tracking under occlusion, tracking for surveillance, single and multi-view tracking, etc. The review search also found use of multi-camera tracking in the medical sector, such as multi-camera tracking systems for respiratory motion tracking. Research on this field has not been explored much and this holds possibilities for further work.**

Keywords—Multi-camera tracking; Motion estimation; FPGA; Respiratory motion tracking

I. INTRODUCTION

One important area of camera tracking systems is to focus on detection of objects, people and vehicles. Surveillance of objects like people in airports to vehicles in inter-city junctions, is done with camera tracking systems. Outdoor tracking such as for vehicles, pedestrians, sports, etc. and indoor monitoring of people, patients in hospitals, etc. are done with single or multi-camera tracking systems.

When multiple cameras are focused on a common field of view (FOV) the overlapping of the images obtained needs to be taken into account. Non-overlapping images are obtained when cameras have separate field of views. Occlusion occurs when the object being tracked is hidden or occluded by another object. Occlusion can be static when the object blocking the target is static and dynamic if both are moving like pedestrians blocked by moving cars.

Multi-camera object or people tracking requires real-time video processing, tracking objects between overlapping and non-overlapping camera views, considering occlusion, fast computation, etc. [1, 2, 3, 4, 5, 6, 7].

Object and people tracking with multi-camera system, multimedia communication such as video conferencing, or even medical imaging - all sectors of real-time video image processing require video compression techniques. Efficient Motion Estimation (ME) and Motion Compensation (MC) algorithms and their successful hardware implementation is the key to video compression [7].

Among the many steps that allow the encoding of digital video sequences with high efficiency, ME is considered the most complex step [8]. Work on developing and implementing efficient ME and MC algorithms for multi-camera systems is ongoing. The low power consumption yet high speed of Heterogeneous Reconfigurable Devices such as Field Programmable Gate Arrays (FPGA) can be suitable for implementation of real-time optic flow computation [9] of multi-camera systems. The struggle is with the efficient use of limited memory and resources of reconfigurable devices.

A study to discover the current state-of-the-art research into multi-camera motion tracking is done in this review paper. The search explores the different purposes behind multi-camera tracking and the techniques used. It explores areas which have been focused on more and which have been neglected and have further research scope. The search yields there have been much study into surveillance with multi-camera tracking systems, but less in health care sector. There is possibility of using multi-camera tracking systems in respiratory motion tracking using extracorporeal (non-invasive) markers [10].

Also, most of the motion optimization techniques have been based on the common block matching algorithms [7]. However, adaptive low-complexity global motion estimation algorithms using logic operations instead of complex arithmetic calculations have also been developed [7, 11, 12]. Implementing these onto reconfigurable devices like FPGAs holds possibility for more efficient memory use and faster computation. But not enough research has been done on these low-complexity algorithms.

Section II of this paper explores the areas of multi-camera motion tracking used in previous researches. Section III reviews the motion optimization techniques that have been used and also the implementation of these on to heterogeneous reconfigurable devices. The fourth section discusses a possible use of multi-camera tracking systems in the medical sector. The fifth and final section suggests possibilities of implementing low-complexity algorithms on to reconfigurable devices for new areas of multi-camera systems, e.g. respiratory motion tracking with multi-camera systems.

II. MULTI-CAMERA TRACKING SYSTEMS

Object or target tracking can be broadly categorized under: Surveillance, Health and Ecological surveys. This

Funding provided by Ministerio de Economía, Industria y Competitividad, Gobierno de España, PLATINO Project (TEC2017-86722-C4-4)

paper focuses on how multiple camera tracking in these cases and the search result shows multi-camera motion tracking has been mainly focused on people and vehicle tracking in indoor and outdoor conditions [1, 5], tracking under occlusion [13], tracking for surveillance [1, 3, 5], single and multi-view tracking [2, 10]. Researchers explored common field of view with multi-camera for single view tracking for occlusion [10]; Inter-camera object tracking (ICT) relying on Single camera object tracking (SCT) [2, 14]; Multi-view image surveillance [1]; Distributed collaborative cameras for multiple human tracking [9]; Uncalibrated cameras for object tracking [15], etc.

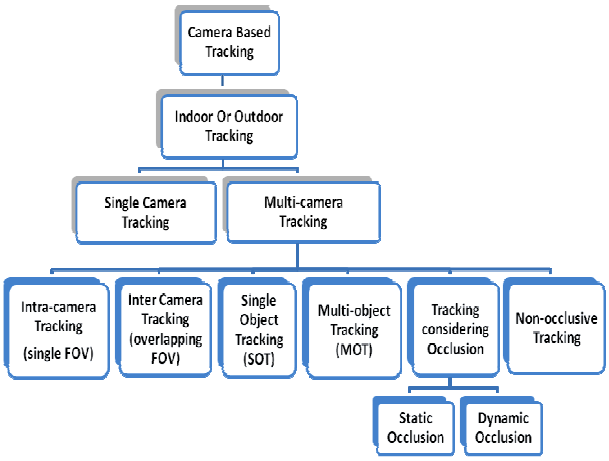


Fig. 1. Conditions under which multi-camera tracking systems have been explored.

Figure-1 shows the range of conditions under which multi-camera tracking systems have been usually explored. The tracking systems can be for indoor or outdoor conditions; they can be for Intra-camera Tracking which is for single Field of View (FOV) and for Inter-camera Tracking (overlapping FOV); multi-camera tracking systems can be for Single Object Tracking (SOT) or for Multi-object Tracking (MOT); they can also consider both static and dynamic occlusion, etc. [15, 16]. The search shows researches into multi-camera tracking systems have taken one or multiple of these conditions under consideration.

According to [16] at least 70% of current MOT research efforts are devoted to pedestrians. Pedestrian and vehicle tracking or surveillance related research has been found to be the main focus area involving multi-camera tracking systems in this review.

Figure-2 reproduced from [14] shows some technologies in intelligent multi-camera video surveillance, where the arrows indicate the information flow. The different views from multi-camera systems are mapped to a single coordinate system by Multi-camera Calibration; overlapping camera views, transition time between camera views, etc. are determined by Camera Topology; Object Re-identification matches image regions in different camera views; Multi-camera Tracking tracks objects across different camera views; and Multi-camera Activity Analysis recognizes activities of different categories [14]. Calibration and topology problems are often jointly solved, as sometimes calibration depends on topology information [14].

More cameras in a multi-camera surveillance system also mean more complexity in the topology of the network. The system needs to consider different fields of view, blind areas between multiple cameras, occlusion handling, changes in illumination, structural changes in the scene, etc. [14].

Inter-camera tracking becomes more challenging as it needs to predict the spatio-temporal information of objects across different camera views and consider changes in the appearance of objects because of camera settings, viewpoints, etc. [14].

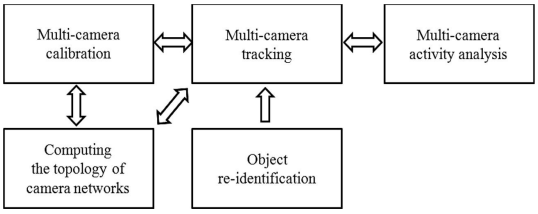


Fig.2. Some technologies in intelligent multi-camera video surveillance. The arrows indicate the information flow between different modules (Wang, 2013). The figure is reproduced from [14].

III. MOTION OPTIMIZATION ALGORITHMS USED

Researchers used different filter techniques such as - particle filters, collaborative filters, Kalman and Extended Kalman filters, Sequential camera filter, etc. and object mapping, color histogram tracker, etc. [4, 16] for object tracking.

For real-time, high definition motion vector determination Block Matching Algorithms (BMA) like Full Search (FS) Algorithm, Three Step Search (TSS), New Three Step Search (NTSS), Diamond Search (DS), etc. have been found to be repeatedly used by researchers [17, 18] in Motion Estimation and Motion Compensation. Sum of Absolute Differences (SAD), Sum of Squared Differences (SSD), modified SAD [19, 20, 21], etc. are also common approaches.

Block matching algorithms divide each frame into a matrix of 'macro blocks' (MB), then compare the location of each MB of the current frame relative to a reference frame, and give the motion displacement between MBs by minimizing a cost function which gives the disparity between them. Matching of one macro block to another depends on cost functions like Mean Absolute Difference (MAD), Mean Squared Error (MSE), etc. [22].

The multiplications associated with these algorithms cause delay in the processing speed [23, 25]. Efficient use of resources in reconfigurable devices remains a problem as well [20]. Some of the works offer higher speed with less computation, but they lack a detailed comparison of efficiency with other work [19].

Implementation onto reconfigurable devices like FPGAs have been suggested in some works and some of these were also carried out [17, 19, 20, 21, 23, 24, 25]. A few performed a comparison between different algorithms and FPGA implementations [18, 24].

Block matching algorithms require complex calculation and use most of the memory and resources of the reconfigurable devices [18, 20, 23]. However, this search into the state-of-the-art also yields that adaptive low-complexity global motion estimation algorithm using logic operation instead of complex arithmetic calculation has been developed [7, 11]. The proposed algorithms reduce the complexity of multiplication and most of the calculation can be replaced by LOGIC operations (e.g. XOR, NAND, NOR,

etc.). A similar algorithm has also been implemented onto an FPGA [12]. The FPGA implementation of a one-bit-per-pixel image registration algorithm proposed in [12] used an Adaptive Low-complexity Algorithm (ALC) developed by the authors. The algorithm was implemented on to a Spartan 3 FPGA, where only 21% of resources were used. It reached 25 frame rates but holds possibility for a maximum frequency of 150 MHz with a potential 75 frame rate. Such low-complexity motion estimation algorithms reduce the memory usage in reconfigurable devices, and calculations can also become faster. But not enough research has been done on these algorithms and logical operations, and this area holds more possibilities to explore.

IV. MULTI-CAMERA TRACKING POSSIBILITY IN MEDICAL SECTOR

While multi-camera tracking is being widely used for surveillance of people and vehicles, this research has found that an area of application for multi-camera tracking system not much explored is the healthcare sector. Monitoring of patients is one of the applications, which again falls into the subcategory of surveillance.

Figure-3 highlights the areas of more research possibilities in the health sector with multi-camera systems. Researches in monitoring patient condition or remote health monitoring with multi-camera system have been done [26, 27] as well as single-camera system for wearable video monitoring [28] and patient associated motion detection with Microsoft Kinect™ camera [29].

Multi-camera systems have also been used to assist in disability, such as - obstacle avoidance, improve wheelchair movements, etc. [30, 31]. Gait analysis for prevention of injuries in elderly people, and to analyze functional decline in neurological diseases or such also have possibilities in single and multi-camera tracking systems [32, 33].

This state-of-the-art search also found use of multi-camera tracking systems in respiratory motion tracking with non-invasive extracorporeal markers [10]. Respiratory motion tracking is important for monitoring tumor positions affected by respiratory motion. For medical imaging (CT Scan, MRI, PET, etc.) and during treatments (radiotherapy, radiofrequency – HIFU, etc.) in organs like liver, lungs, etc. the position of the tumors change with respiration [34] and effective real-time Respiratory Motion Models are needed. Work in this field in the last 15 years involves invasive and non-invasive markers and tracking. A multi-camera vision system is proposed in [10] to track extracorporeal markers. The authors used a modified binocular equation for multi-camera system which can simulate respiratory motion. The 3D coordinates of the markers were calculated with some errors.

Respiratory monitoring with multi-camera system has been researched in pediatric intensive care environment as well [35]. A few of the later researches used Microsoft Xbox Kinect™ cameras for respiratory motion tracking [36, 37], but not much has been explored with multi-camera tracking systems and this area holds possibilities for further work.

V. FUTURE DIRECTIONS

This paper reviews the current state-of-the-art in multi-camera tracking systems, and finds that although much focus has been given to pedestrians and vehicles for surveillance and detection, multi-camera systems can be applied in more research fields such as healthcare sector, smart cities,

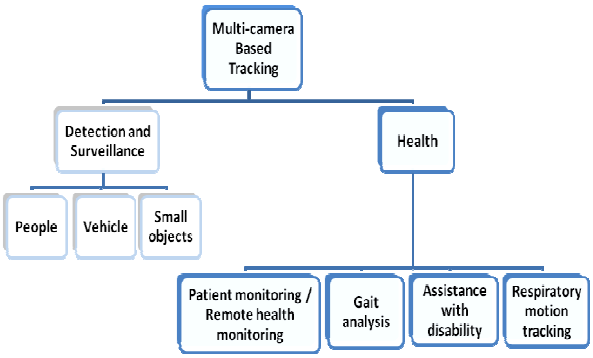


Fig. 3. Areas of further research possibilities with multi-camera tracking systems in the medical sector.

Specifically in the healthcare sector there has been some work done on gait analysis [32, 33] which holds more possibilities for research. Some other fields with further research possibilities are assistance systems for people with disabilities [30, 31], obstacle detection for elderly people living alone at home or for patients with degenerative motor conditions, etc.

One interesting area of research in the medical sector is respiratory motion tracking with multi-camera system. While research in this field is ongoing [10, 35, 36, 37], there is a possibility of using multiple cameras to track respiratory motion and implementing low-complexity algorithms on to reconfigurable devices.

Implementing different motion optimization algorithms onto reconfigurable devices suffer from the problem of implementing complex arithmetic calculations and the limited memory available in the reconfigurable devices [18, 20, 23]. Often these limit the fast computations required in real time image processing a multi-camera system needs. Proposed low-complexity algorithms [7, 11] may solve this.

Future work possibilities in the field of multi-camera tracking systems may be in implementing these low complexity algorithms in respiratory motion tracking with multi-camera tracking systems.

REFERENCES

[1] J. Black, T. Ellis, and P. Rosin; "Multi view image surveillance and tracking"; Proceedings of the Workshop on Motion and Video Computing, IEEE 2002, DOI: 10.1109/MOTION.2002.1182230

[2] W. Chen, L. Cao, X. Chen, and K. Huang, "A novel solution for multi-camera object tracking", 2014 IEEE International Conference On Image Processing (ICIP 2014), pp. 2329-2333, DOI: 10.1109/ICIP.2014.7025472

[3] B. Deutsch, S. Wenhardt, and H. Niemann, "Multi-step multi-camera view planning for real-time visual object tracking"; K. Franke Et Al. (Eds.): DAGM 2006, LNCS 4174, pp. 536–545, 2006; Springer-Verlag Berlin Heidelberg 2006.

[4] B. Purnama, B. Erfianto, and Y. Hafidz, "On the experiment of multi camera tracking using Kalman filter and FOV lines", 2014 2nd International Conference on Information and Communication Technology, ICoICT 2014, pp. 297-301, DOI: 10.1109/ICoICT.2014.6914082

[5] J. Castañeda, V. Jelaça, A. Frías, A. Pižurica, W. Philips, R. Cabrera, and T. Tuytelaars, "Non-overlapping multi-camera detection and tracking of vehicles in tunnel surveillance", Proceedings - 2011 International Conference on Digital Image Computing: Techniques and Applications, DICTA 2011, pp. 591-596 , DOI: 10.1109/DICTA.2011.105

[6] L. Anuj, M.T.G. Krishna, "Multiple camera based multiple object tracking under occlusion: A survey", International Conference on Innovative Mechanisms for Industry Applications (ICIMIA 2017), pp. 432-437, DOI: 10.1109/ICIMIA.2017.7975652

[7] S Ertürk, "Multiplication-free one-bit transform for low-complexity block-based motion estimation", IEEE Signal Processing Letters, VOL. 14, NO. 2, February 2007, pp. 109-112, DOI: 10.1109/LSP.2006.882088

[8] H. Maich, G. Paim, V. Afonso, L. Agostini, B. Zatt, and M. Porto; "A multi-standard interpolation filter for motion compensated prediction on high definition videos", 2015 IEEE 6th Latin American Symposium On Circuits And Systems, LASCAS 2015 - Conference Proceedings; DOI: 10.1109/LASCAS.2015.7250478

[9] Z. Cai, S. Hu, Y. Shi, Q. Wang, and D. Zhang, "multiple human tracking based on distributed collaborative cameras", Multimedia Tools Applications, 2017; pp. 76: 1941– 1957; DOI: 10.1007/S11042-015-3163-7

[10] L. Ding, H. Zhang, and Y. Xie, "Respiratory motion tracking with a multi-camera vision system", 2013 IEEE International Conference on Medical Imaging Physics and Engineering; DOI: 10.1109/ICMIPE.2013.6864567

[11] M. Haque, M. Biswas, M. Pickering, and M. Frater, "An adaptive low-complexity global motion estimation algorithm", 28th Picture Coding Symposium, DOI: 10.1109/PCS.2010.5702574

[12] A. Nguyen, M. Pickering, and A. Lambert, "The FPGA implementation of a one-bit-per-pixel image registration algorithm", Journal of Real-Time Image Processing Archive, Volume. 11 Issue. 4, April 2016, pp. 799-815, Springer-Verlag New York, Inc. Secaucus, NJ, USA; DOI:10.1007/S11554-014-0420-3

[13] J. P. Batista, "Tracking pedestrians under occlusion using multiple cameras", FCT Project POSI/SRI/34409/1999, A. Campilho, M. Kamel (Eds.): ICIAR 2004, LNCS 3212, pp. 552–562, 2004, Springer-Verlag Berlin Heidelberg 2004.

[14] X. Wang, "Intelligent multi-camera video surveillance: A review", Pattern Recognition Letters, Vol. 34 Issue 1, 2013, pp. 3-19

[15] D. Wedge, A. Scott, Z. Ma, and J. Vendrig, "Object tracking over multiple uncalibrated cameras using visual, spatial and temporal similarities", J. Blanc-Talon Et Al. (Eds.): ACIVS 2010, Part II, LNCS 6475, pp. 167–178, 2010, Springer-Verlag Berlin Heidelberg 2010.

[16] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, and T. Kim, "Multiple object tracking: A literature review"; ArXiv IDarXiv:1409.7618v4, 2017, pp. 1-18.

[17] E. Castillo, C. Cárdenas, and M. Jara, "An efficient hardware architecture of the H.264/AVC half and quarter-pixel motion estimation for real-time high-definition video streams"; 2012 IEEE 3rd Latin American Symposium on Circuits and Systems (LASCAS), pp. DOI: 10.1109/LASCAS.2012.6180302

[18] B. Biswas, R. Mukherjee, and I. Chakrabarti, "An efficient VLSI architecture for motion estimation using New Three Step Search algorithm", TENCON 2014 - 2014 IEEE Region 10 Conference, DOI: 10.1109/TENCON.2014.7022424

[19] M. Mori, T. Itou, M. Ikebe, T. Asai, T. Kuroda, and T. Motomura, "FPGA-sased design for motion-vector estimation exploiting high-speed imaging and its application to machine learning", 2014 RISP International Workshop on Nonlinear Circuits, Communications and Signal Processing, NCSP'14, Honolulu, Hawaii, USA.

[20] E. Alcocer, O. López-Granado, R. Gutiérrez, and M.P. Malumbres, "An FPGA-based integer motion estimator for real time HEVC UHD video encoding", XXIII Seminario Anual De Automatica, Electronica Industrial E Instrumentacion (SAAEI16) 2016, pp-18.

[21] B.M.H. Li and H. W. Leong, "Serial and parallel FPGA-based variable block size motion estimation processors", Journal Of Signal Processing Systems, Vol. 51, pp. 77–98, 2008; DOI: 10.1007/S11265-007-0143-9

[22] W. Hassen and H. Amiri, "Block matching algorithms for motion estimation", 2013 7th IEEE International Conference on e-Learning in Industrial Electronics (ICELIE), DOI: 10.1109/ICELIE.2013.6701287

[23] D. Palomino, F. Sampaio, S. Bampi, A. Susin, and L. Agostini, "FPGA based design for motion vector predicton in H.264/AVC encoders targeting HD1080p resolution", 2012 VIII Southern Conference On Programmable Logic, DOI: 10.1109/SPL.2012.6211785

[24] H. Maich, G. Paim, V. Afonso, L. Agostini, B. Zatt, and M. Porto, "A multi-standard interpolation filter for motion compensated prediction on high definition videos", 2015 IEEE 6th Latin American Symposium on Circuits and Systems, LASCAS 2015 - Conference Proceedings, DOI: 10.1109/LASCAS.2015.7250478

[25] G. Pastuszak, "Architecture design of the H.264/AVC encoder based on rate-distortion optimization", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 25 Issue 11, 2015, pp. 1844-1856, DOI: 10.1109/TCSVT.2015.2402911

[26] O. Gupta, D. McDuff, and R. Raskar "Real-time physiological measurement and visualization using a synchronized multi-camera system", 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 312-319, DOI: 10.1109/CVPRW.2016.46

[27] T. Elgamal and K. Nahrstedt, "Multicamera summarization of rehabilitation sessions in home environment", Proceedings of the 25th ACM international conference on Multimedia, 2017, pp. 1381-1389, DOI: 10.1145/3123266.3123387

[28] R. Mégret, V. Dovgalecs, H. Wannous, S. Karaman, J. Benois-Pineau, E. El Khoury, J. Pinquier, P. Joly, R. André-Obrecht, Y. Gaëstel, and J-F. Dartigues, "The IMMED project: wearable video monitoring of people with age dementia", The IMMED project. Proceedings of the International Conference on Multimedia - MM '10 (2010), DOI: 10.1145/1873951.1874206

[29] L. Liu and S. Mehrotra, "Patient associated motion detection with optical flow using Microsoft Kinect V2", Proceedings of the Second IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies, 2017, pp. 274-275, DOI: 10.1109/CHASE.2017.99

[30] I. R. Nourbakhsh, D. Andre, C. Tomasi, and M. R. Genesereth, "Mobile robot obstacle avoidance via depth from focus", Robotics and Autonomous Systems, Vol. 22, 1997, pp. 151-158.

[31] V.K. L. Ha, T. N. Nguyen, and H.T. Nguyen, "Real-time video streaming with multi-camera for a telepresence wheelchair", 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2016, DOI: 10.1109/ICARCV.2016.7838749

[32] X. Gu, F. Deligianni, B. Lo, W. Chen and G.Z. Yang, "Markerless Gait analysis based on a single RGB Camera", 2018 IEEE 15th International Conference on Wearable and Implantable Body Sensor Networks (BSN), 2018, pp. 42-45, DOI: 10.1109/BSN.2018.8329654

[33] M. S. Shafiq, S.T. Tümer, H.C. Güler, "Marker detection and trajectory generation algorithms for a multicamera based gait analysis system", Mechatronics, Volume 11 Issue 4, 2001, pp. 409-437.

[34] J. McClelland, D. Hawkes, T. Schaeffter, and A. King, "Respiratory motion models: A review", Medical Image Analysis October 2012; DOI: 10.1016/j.media.2012.09.00

[35] H. Rehouma, R. Noumeir, W. Bouachir, P. Jouvet, and S. Essouri, "3D imaging system for respiratory monitoring in pediatric intensive care environment", Computerized Medical Imaging and Graphics, Vol. 70, 2018, pp. 17-28.

[36] S.H. Lim, E. Golkar, and A.A.R. Ashrani, "Respiratory motion tracking using the KinectTM camera", 2014 IEEE Conference on Biomedical Engineering and Sciences; DOI: 10.1109/IECBES.2014.7047619

[37] M. Samir, E. Golkar, and A.A.R. Ashrani, "Comparison between the KinectTM V1 and KinectTM V2 for respiratory motion tracking", 2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA); DOI: 10.1109/ICSIPA.2015.7412180

[38] M. Kamezaki, J. Yang, H. Iwata, S. Sugano, "An autonomous multi-camera control system using situation-based role assignment for tele-operated work machines", 2014 IEEE International Conference on Robotics and Automation (ICRA),DOI: 10.1109/ICRA.2014.6907739

New Approach of Pipelined Architecture of SVM Classifier for ASR System in FPGA Implementation

Gracieth C. Batista, Duarte L. Oliveira, Osamu Saotome, Leonardo Romano
Electronic Engineering Division – Technological Institute of Aeronautics – IEEA – ITA
São José dos Campos – São Paulo – Brazil
{gracieth, duarte, osaotome}@ita.br, leoroma@uol.com.br

***Abstract* — In this paper, it is proposed an pipelined architecture for classification of Support Vector Machine (SVM) algorithm for an Automatic Speech Recognition application that it has a use of two different gated-clock control circuits to synchronize its pipelined datapath. About these control circuits, one is based on Extended Burst Mode specification and the other one is based on transparent latches. In addition, the training data is optimized by Particle Swarm Optimization (PSO) algorithm that it is applied before SVM training, i.e., the training data is obtained from a hybrid training (PSO-SVM training) algorithm. Then, the tests are from 60 speeches and 20 speakers, so it is a diversified dataset of test. The training/learning part was implemented in software (Matlab) and the testing/classification part was implemented in FPGA, described in VHDL. The main goal here was to obtain the fastest response time which was about 50 – 500 GOPS of throughput; the accuracy in recognition success rate was other preoccupation and it was very successful, 99% of success; the operating frequency of clock was 100 – 500MHz.**

***Keywords*—FPGA Machine Learning Implementation; SVM classification; Pipelined Architecture; Speech Recognition.**

I. INTRODUCTION

Pattern recognition algorithms have been used commonly in order to accelerate manual processes and/or to automate entirely most of these. Speech recognition is one of patterns recognition area and it is very important and it is used in many devices currently [1]. In those kind of algorithms, there are applications of Neural Networks, Deep Learning, Machine Learning (ML), Heuristic methods, etc; among them, Support Vector Machine (SVM) model is a technique of supervised ML which is robust to applications with many different patterns to be recognized and it presents efficient classification. It is used in regression and prediction algorithms [2]. Many techniques are combined to SVM training in order to decrease its processing time and training data [3], such as optimization algorithms.

SVM model is computationally expensive and time-consuming especially for large-scale problems, which raises a vital need for acceleration. While software implementations of SVM produce high accuracy rates, they cannot efficiently meet real-time embedded systems constraints. In such embedded real-time applications, special dedicated hardware architectures are required to meet constraints like limited resources utilization and low power consumption [7]. Reconfigurable hardware is a promising solution for speeding up operations and it provides high performance at low cost and low power consumption. Yet, knowing that training phase has many computations, it is more convenient to do that part in software and in hardware, just the classification phase; such assertion can be analyzed and reaffirmed through [13] where it is found many types of FPGA implementations of SVM algorithm.

Song et al. in [17] implemented a linear and non-linear SVM classifier however, not automatic because its data entrance are made manually, neither optimized because its training data are all from data pre-processing part then it took much time to do the whole SVM training phase, and it is a combinational circuit that it costs less power consumption but worse response time. On the other hand, Batista and Silva in [12] implemented PSO-SVM hybrid training algorithm and SVM classification however without hardware application, i.e., the whole recognition system was implemented in software. The presented Automatic Speech Recognition (ASR) system has the following features: multi-class (30 classes), application of RBF kernel function and 3 stages in its pipelined datapath.

In this paper, it is proposed a new basic gated-clock architecture to implement synchronous pipeline systems that is insensitive to global clock. One of its proposed architecture has as main characteristic, when compared with other pipeline architectures, the use of a simple asynchronous control to perform the operations (XBM control circuit) different from basic pipelined architectures. Basic pipelined architectures only activate the registers when there is valid data to store in the different stages, it indicates when there is valid data in the output and the input accepts data that arrives at a frequency unrelated to the operation frequency of the pipeline and it may require several clock cycles or even a fraction of clock cycles.

This paper is organized in four sections: Section II presents a background of concepts about SVM, PSO and XBM; Section III has the details of hybrid training phase (PSO-SVM training algorithm) processed in Matlab software; Section IV has the details of SVM classification phase (pipelined architecture in VHDL description and implemented in FPGA); Section V has the systems results; and finally, Section VI has conclusion where there are the vantages and advantages results.

II. BACKGROUND CONCEPTS

A. SVM algorithm

SVM was introduced by Vapnik and Chervonenkis [8,9] and since then, it has been applied in many areas [14,15]. SVM method is composed by two phases: training/learning and testing/classification. As being a dichotomic algorithm, there are ways to use it for more than two classes (patterns), known as multi-class SVM (i.e., non-linear classification).

B. PSO algorithm

Particle Swarm Optimization (PSO) is a bio-inspired algorithm that calculates the best position of a data population in coordinates space [6] and it was introduced by Kennedy and Eberhart [4,5].

C. XBM circuit

The XBM is originated from Burst Mode specification that also it is represented by a diagram of state transitions where transitions can occur for single or multiple input changes. It is necessary to define the initial state, each state transition is represented by an arc in the diagram and it is labeled by a set of input signals and/or output signals. These sets are called input/output burst. Based on a state diagram, XBM shows to be very familiar to designers, it inherits all the characteristics of BM specification and incorporates the "level conditional signals" and "direct don't-care signals" [21].

III. PSO–SVM HYBRID TRAINING IN SOFTWARE

The banks are composed from 20 speakers (male and female), each of one spoke 30 speeches (being 20 voice commands and 10 digits – zero to nine) where each speech is spoke 20 times. Those banks are distinct, that is, one bank is for training and another is for classification/test.

The hybrid training (PSO-SVM algorithm) is initialized from parameters extracted from the pre-processed speech signal. This pre-processing was made from data acquisition of the distinct voice banks, as it follows of spectral analysis of speech signals, then a Hamming windowing application and Mel-Frequency Cepstral Coefficients (MFCCs) are obtained from a series of calculations; finally, in order to reduce those coefficients, the Discrete Cosine Transform (DCT) is used. Parameters from this pre-processing are re-allocated in matrices of 2x2.

Those 2x2 matrices are entered in PSO algorithm at the same time inside of the CT_{Nx2} matrix as it follows in Equation1.

$$g(\alpha,\beta) = (\alpha - a)^2 + (\beta - b)^2 \qquad (1)$$

where α and β are pairs from the CT matrix that it is transformed in a only one pair at the end of PSO process, *a* and *b* are points that are localized in the center of each class, *a* related to the *x* axis and *b* related to *y* axis [6].

Once the optimized CT matrix is obtained, then the SVM multi-class training (one vs. all) [11] is realized from Radial Basis Function (RBF) kernel [11] with 0.9 sigma constructing an optimum hyperplane that obeys to Equation 2.

$$w^T x + b = 0 \qquad (2)$$

where *w* is a matrix of adjustable weight values, *b* is a matrix of bias values and *x* is a matrix of entrance data values of SVM algorithm.

SVM is based on Statistical Learning Theory and Risk Functional/VC dimension concepts for the purpose of finding the best machines performance [2].

In the end, the one Support Vectors (SVs) matrix (30x2 dimension), thirty Lagrange Multipliers matrices (30x2 dimension each) and one Bias matrix (30x1 dimension) are obtained.

IV. SVM CLASSIFIER IN HARDWARE

In hardware application, the flowchart of Figure 2a) was generated from the SVM classifier algorithm in pipelined configuration.

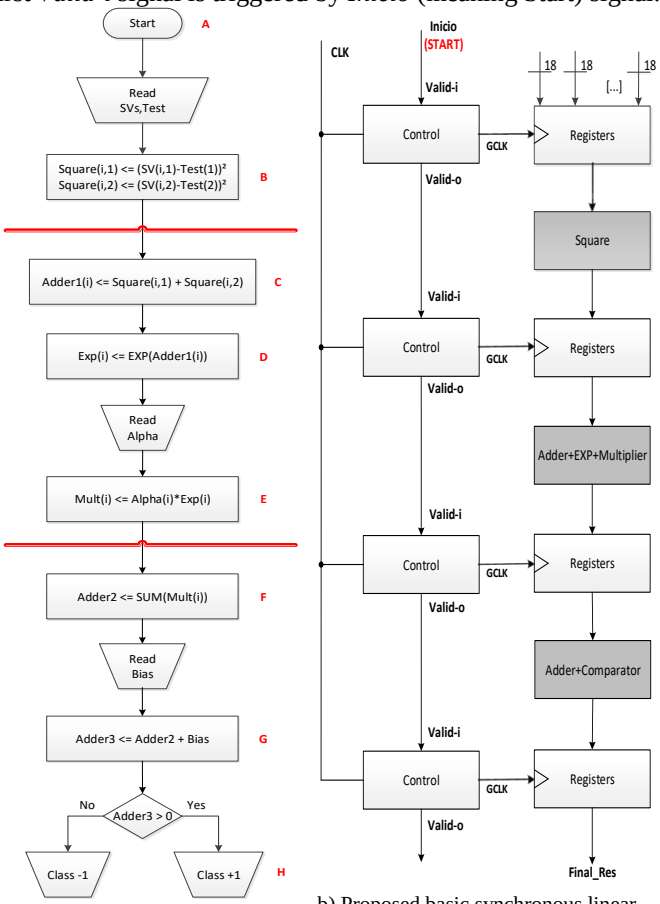
In order to do the VHDL description, the datapath was generated for 30 classes according to flowchart of Figure 2a).

The *ieee_proposed.fixed_pkg.all* package was used for converting the floating point to 18 bit fixed point format [20]. The *EXP()* component is a look up table described by Equation 3.

$$\text{EXP}\left(-(SV - Test)^2 / 2 * \sigma^2\right) \qquad (3)$$

where $\sigma = 0.9$, *SV* are the *Support Vectors* values and *Test* are values from the speech signals parameters of voice bank test.

The proposed basic pipeline architecture insensitive to latency is composed of processing modules as shown in Figure 2b). Where it is possible to observe the datapath of the SVM classifier showing the connections among the SQUARE, EXP, MULTIPLIER, COMPARATOR and ADDER components; there are also the SVs, Test and Alpha (Lagrange Multipliers coefficients) matrices as data entrances of 18 bits on the top. All these matrices came from the training processed in Matlab software. The registers are based on D flip-flop, the control in each pipeline register has two types in its configuration and the first *Valid-i* signal is triggered by *Inicio* (meaning Start) signal.



a) Flowchart of the SVM classifier divided in 3 parts (A and B / C, D and E / F, G and H) according to its pipelined architecture of 3 stages.

b) Proposed basic synchronous linear pipeline: it only storages data when there is a valid signal in the first control circuit independently from clock.

Fig. 2. Details of the proposed pipelined architecture

The first configuration is about asynchronous controller circuits to perform synchronization between the global clock signal and a signal sensitive to *Valid-i* signal level in the

specification XBM shown in Figure 3. It is composed of the following signals: *Valid-i* (valid data at the stage input - request-store), *Valid-o* (data valid for the next stage, a signal to ask for requesting to store), *CLK* (clock) signal and *GCLK* (gated-clock signal) signal controlled by valid data in gated stores. Then, Figure 4 shows the logic circuit of the asynchronous control circuit that it was synthesized by the 3D tool [16].

Finally, the second configuration of control is a version based on transparent latches that is presented and its logic circuit is presented in Figure 5.

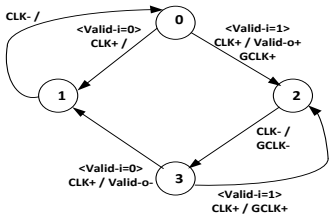


Fig. 3. XBM specification: control for elastic synchronous pipeline with only one valid signal to control GCLK output.

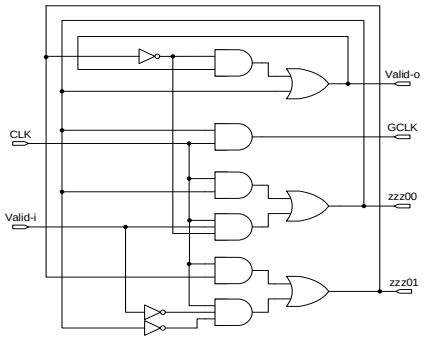


Fig. 4. Gated-Clock XBM Control: logic circuit.

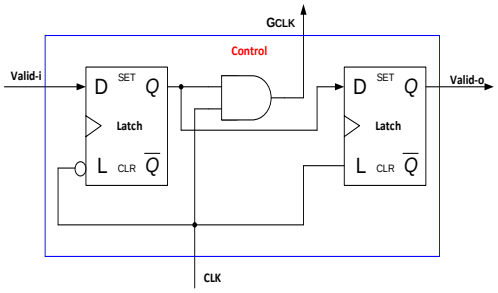


Fig. 5. Gated-Clock Control: logic circuit.

V. RESULTS AND ANALYSIS

The tests were made from a dataset size of 60 speeches from 20 speakers (male and female). The proposed architecture was targeted in Altera's Cyclone II EP2C70F896C6 with operating frequency of 100 –500MHz (clock frequency), but it can be changed to any other targeted device. In this paper, it is made comparisons of four versions of the SVM classifier, as it follows:

1. Batista and Silva's implementation processed in Matlab software [12];
2. Song et al.'s implementation in Combinational Circuit (CC) processed in FPGA implementation [17];
3. Proposed pipelined architecture with control based on XBM specification processed in FPGA implementation;
4. Proposed pipelined architecture with control based on transparent latches processed in FPGA implementation.

About first comparison, the training algorithm execution in software was the same that Batista and Silva [12] made, perhaps in Intel Core i7-6500U CPU @ 2.50GHz – 64 bits Operating System, just with the difference of sigma RBF kernel function equal to 0.03. And in this proposed recognition system, it was used a sigma equal to 0.9 what should provide worse results in classification phase, theoretically; yet it was obtained a recognition success rate of 98.5 to 100% and response time frequency of 150 to 250MHz. In the software version in [12], it is presented a recognition success rate of 89 to 92% and response time of 15 to 170Hz.

About second comparison, Table I lists the difference between results of Song et al.'s implementation and the same proposed architecture, i.e., a combination circuit version without control circuit. Where there is no comparison in dissipation power area because in Song et al's paper [17], it is not cited it. So, it is possible to analyze difference among RBF kernel function sigma, recognized patterns/classes, number of classified/tested models, amount of classified/tested samples, registers quantity, maximum operation frequency of clock and recognition success rate parameters.

In Table II is listed the ModelSim ALTERA simulation results with both control circuits applications from lowest circuit configuration where it is considered the worst circuit conditions like critical path, highest temperature, etc. Then, it is possible to observe the difference of throughput, LUTs, recognition success rate, power consumption and maximum operation frequency parameters.

The result of throughput parameter of all implementations of SVM classification phase is on Table III, i.e., the SVM classifier processed in Matlab software [12] (item 1 of the table), implementation of Song et al. architecture [17] (item 2 of the table) and two proposed pipelined architectures (items 3 and 4 of the table).

TABLE I. DIFFERENCE BETWEEN THE COMPARED ARCHITECTURES

FEATURES	SONG'S	CC CONFIGURATION
SIGMA RBF KERNEL FUNCTION	0.6	0.9
CLASSES (PATTERNS)	10	30
MODELS (SPEAKERS AMOUNT)	4	20
TESTING SIZE	40	60
REGISTERS	0	1
MAXIMUM OPERATION FREQUENCY OF CLOCK (MHZ)	250	240
RECOGNITION SUCCESS RATE (%)	97 – 100	98.5 – 100

TABLE II. RESULTS OF THE PROPOSED PIPELINED ARCHITECTURES

FEATURES	XBM - CONTROL CIRCUIT	LATCH - CONTROL CIRCUIT
THROUGHPUT (GOPS)	50	500
LUTS	6,198	6,198
RECOGNITION SUCCESS RATE (%)	98.5 – 100	98.5 – 100

DYNAMIC POWER DISSIPATION (MILIWATTS)	174.15	224.92
MAXIMUM OPERATION FREQUENCY OF CLOCK (MHZ)	500	250

TABLE III. RESULTS OF THROUGHPUT BETWEEN IMPLEMENTATIONS OF SVM CLASSIFERS

	THROUGHPUT
1) MATLAB SOFTWARE	170 OPS
2) SONG ET AL' IMPLEMENTATION	150 MOPPS
3) PIPELINED ARCHITECTURE IMPLEMENTATION (ASSYNCHRONOUS CONTROL CIRCUIT - XBM)	50 GOPS
4) PIPELINED ARCHITECTURE IMPLEMENTATION (SYNCHRONOUS CONTROL CIRCUIT - LATCHES)	500 GOPS

VI. CONCLUSION

Then, from comparison between processes in PC and FPGA, it is possible to observe that this system implemented in FPGA is a way better in response time and success rate as it should be.

From comparison between Song's architecture [17] and the proposed one in combinational circuit mode, it is possible to observe that the proposed architecture is better, because of the following items:

- Sigma RBF kernel function is higher in the proposed architecture yet it provides even better classification results, in reason of the exponential calculation which has the same 14 bits of precision in its decimal part;
- It gives a system recognition more applicable because it has more patterns to be recognized, i.e., more understandable voice commands;
- The operating frequency of clock is worse than in Song's architecture in reason of using registers, such use is necessary due to the proposal fact that it presents automatic entrance of test dataset;
- It is an automatic system because training dataset is applied in the classification/test architecture (in FPGA) and not manually as it is in Song's architecture (through a testbench file, for example).

Besides, it is observed in pipelined architecture from comparison between both control circuits that asynchronous circuit (XBM specification) presented worse throughput; however it showed better performance in power consumption, latency and operating frequency rate. It is worth mentioning that asynchronous applications in the area are innovations in literature, a survey of state-of-art implementations of SVM model reassures that [13].

A huge difference between Song et al' implementation results and the proposal pipelined architecture one is analyzed and, it is easily observed that pipeline mode presented the best recognition system. This assertion is made upon the analysis of response time and throughput parameters, number of classification patterns and training models, being all these features very important to any speech recognition system.

In brief, the proposed pipelined architecture is a specific idea of implementation for patterns recognition system, very simple, optimized and faster in response time. For future work, it is intended to do more different architecture configurations for area, performance and response time gains.

REFERENCES

[1] C. Burges, "A tutorial on support vector machines for pattern recognition," Data Mining and Knowledge Discovery, v.2, n.2, pp. 121-167, June 1998.

[2] S. Haykin, "Neural Networks: Principles and practice", 2nd ed., [S.l.]: Bookman, 2001.

[3] B. Kanisha, S. Lokesh, P. M. Kumar, P. Parthasarathy and G. C. Babu, "Speech recognition with improved support vector machine using dual classifiers and cross fitness validation," IEEE Transactions on VLSI Systems, vol. 6, no 4, pp.643-655, Dec.1998.

[4] J. Kennedy and R. Eberhart, "Particle swarm optimization", *Proceedings of the IEEE Conference on Neural Networks*, v. 1, n. 1, pp. 1942-1948, 1995.

[5] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm", Proceedings of IEEE Conference on Systems, Man and Cybernetics, v.1, n.1, 1997.

[6] A. Lazinica, "Particle Swarm Optimization", 1st ed. [S.l.]: In-Tech, 2009. ISBN 978-953-7619-48-0.

[7] Y. Yuan, K. Viruakshappa, Y. Jiang and E. Oruklu, "Comparison of GPU and FPGA based hardware platforms for ultrasonic flaw detection using support vector machines" *Proceedings of the IEEE International Ultrasonics Symposium*, pp. 217-228, November 2017.

[8] V. Vapnik, "The nature of statistical learning theory", 2ed. [S.l.]: Spring-Verlag, 2000. (Statistical for Engineering and Information Science).

[9] V. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of their probabilities to events", Springer-Verlag Berlin Heidelberg 2013, pp. 7-12, October 1967.

[10] P. P. Chu, "FPGA Prototyping by VHDL Examples," A John Wiley & Sons Inc., 2008.

[11] G. C. Batista and W. L. S. Silva, "Using support vector machines and two dimensional discrete cosine transform in speech automatic recognition", *Proceedings of the IEEE International Joint Conference on Neural Networks*, July 2015.

[12] G. C. Batista and W. L. S. Silva, A. G. Menezes, "Automatic Speech Recognition Using Support Vector Machine and Particle Swarm Optimization", *Proceedings of the Symposium Series on Computational Intelligence*, November 2016.

[13] S. M. Afifi, H. G. Hosseini and R. Sinha, "Hardware Implementation of SVM on FPGA: A State-of-the-Art Review of Current Practice", International Journal of Innovative Science, Engineering & Technology, Vol. 2 Issue 11, Nov. 2015.

[14] M. Papadonikolakis and C. S. Bouganis, "A scalable FPGA architecture for non-linear SVM training" *Proceedings Int. Conf. FPT Technol.*, pp. 337-340, December 2008.

[15] D. Anguita, A. Boni and S. Ridella, "A digital architecture for support vector machines: Theory, algorithm and FPGA implementation", Personal and Ubiquituous Computing, Springer-Verlag London Ltd., April 2018.

[16] Goldberg, David E., Genetic algorithms in search, optimization and machine learning, Boston, Addison-Wesley, 1989 p. 412.

[17] X. Song, H. Wang and L. Wang, "FPGA Implementation of a Support Vector Machine based Classification System and its Potential Application in Smart Grid", *Proceedings of the IEEE 11th International Conference on Information Technology: New Generations*, pp.397-402, June 2014.

[18] D. Syu, S. Syu, S. Ruan, Y. Huang and C. Yang, "FPGA implementation of automatic speech recognition system in a car environment", *Proceedings of the IEEE 4th Global Conference on Consumer Electronics*, pp. 485–486, February 2016.

[19] M. Lee, K. Hwang, J. Park, S. Choi, S. Shin and W. Sung, "FPGA-based Low-power Speech Recognition with Recurrent Neural Networks", IEEE International Workshop on Signal Processing Systems, September 2016.

[20] D. Bishop, "Fixed and floating point packages for VHDL 2005", Eastman Kodak Company, Rochester, NY, 2005.

[21] K. Y. Yun e D. L. Dill, "Automatic Synthesis of Extended Burst-Mode Circuits: Part I (Specification and Hazard-.Free Implementation) and Part II (Automatic Synthesis)", IEEE Trans. on CAD of Integrated Circuit and Systems, Vol. 18:2, pp. 101-132, Feb. 1999.

Author Index

XILINX
ALL PROGRAMMABLE™

SATELLOGIC

SYNOPSYS®
Silicon to Software™

UNTREF

CADIEEL

INTI

UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

UAM
UNIVERSIDAD AUTONOMA DE MADRID

Universidad Nacional de San Luis

CAS
IEEE CIRCUITS AND SYSTEMS SOCIETY
ITCR Student Chapter

UNIVERSIDAD FASTA

utn bhi
UNIVERSIDAD TECNOLOGICA NACIONAL
Facultad Regional Bahía Blanca

AGENCIA

CIC COMISIÓN DE INVESTIGACIONES CIENTÍFICAS
Ministerio de Ciencia, Tecnología e Innovación

DIEC

UNS