

An Experimental Comparison of Arduino IDE Compatible Platforms for Digital Control and Data Acquisition Applications

O. E. Amestica, P. E. Melin, C. R. Duran-Faundez, G. R. Lagos

Abstract— This work studied different digital embedded platforms which can be programmed using Arduino IDE software and which can be used for digital control and data acquisition applications. Specifically three kind of Arduino Board (Arduino UNO, Arduino Mega y Arduino Due) and two ESP based board (ESP8622 and ESP32) are evaluated and compared. The comparison is based on the execution time of mathematical operations and a function required for data acquisition or digital control and includes (i) digital ports writing, (ii) analog signal acquisition, (iii) execution of mathematic operation in integer and float format, and (iv) the execution of the data processing code. The execution time is measured by using a methodology which is simple to implement for any other digital device. Because of each platform uses its own Arduino DUE library to generate the assembly code for its own microcontroller, the instructions adopted for both generating the actions to measure and generating the testing signals are written using Arduino IDE commands and using instructions that any entry-level user may adopt. In conclusion, from the experimental results it is observed the high difference in the execution time of the different platforms, especially between ESPx boards and Arduino boards.

Keywords— *Open Hardware, Arduino, ESP8622, ESP32, Internet of Thing*

I. INTRODUCCIÓN






Los sistemas de control electrónicos basados en plataformas electrónicas digitales son una tecnología madura y de amplia aplicación en ambientes industriales, donde hoy por hoy es normal encontrar sistemas de control digital del tipo Programmable Logic Controller (PLC) o sistemas más complejos, como los sistemas Distributed Control Systems (DCS) [1]. Estos dispositivos cumplen con las características de tipo industrial, como por ejemplo resistencia a polvo y humedad. Sin embargo, debido a sus capacidades de procesamiento y/o costos, no son utilizados en sistemas que requieren alta tasa de muestreo [2]. Por otro lado, en aplicaciones del tipo Custom-Device, y más aún en las etapas de investigación y desarrollo de nuevos equipos y aplicaciones de control, es común el uso de dispositivos embebidos para realizar acciones de monitoreo y control. A diferencia de los procesos industriales, y principalmente debido a las condiciones de los procesos de desarrollo, donde se requiere

una alta velocidad de procesamiento y bajo costo, típicamente el uso de dispositivos digitales embebido reemplaza el uso de PLCs. Así, durante la década del 90 y principio de los 2000 existe una fuerte tendencia en el uso de dispositivos Digital Signal Processor (DSP) montados en equipos computacionales, para posteriormente comenzar el uso de estos dispositivos en formato stand-alone [3]. Estos últimos son altamente apreciados en los ambientes de investigación y desarrollo, a pesar de la aparición de los dispositivos Field-Programmable Gate Array (FPGA), cuyo desempeño es mejor pero el costo de desarrollo es mayor, o de equipos como las DSpace (www.dspace.com), cuyo costo de desarrollo es significativamente menor gracias a la integración con softwares como Matlab®, Labview® o MS Excel®, pero con un costo económico tan alto que se hace impensado utilizar estos equipos fuera de los laboratorios de investigación y desarrollo. Lo anterior se ve reflejado en la gran cantidad de artículos técnicos publicados, donde estrategias de control o la adquisición de datos se realiza con estos dispositivos.

En 2005 comienzan aparecer plataformas de desarrollo denominadas Hardware Libre [4]. Estas plataformas integran un dispositivo microcontrolado, típicamente basado en un procesador de 8 bits (ATmega168 en sus inicios, y ATmega328 en la actualidad) con un software de desarrollo fácil de usar por cualquier persona con conocimientos de lenguaje C, y con una gran cantidad de documentación y librerías desarrolladas por los usuarios de la plataforma y distribuida libremente por Internet. Esto permitió que una gran cantidad de personas pudieran integrar dispositivos embebidos en sus proyectos sin la necesidad de tener conocimientos profundos en la utilización de sistemas embebidos o conocimiento en electrónica. Hoy en día es común que alumnos de pregrado, o incluso de enseñanza escolar, sepan utilizar estos dispositivos [6]-[9]. Por otro lado, la plataforma de desarrollo Arduino IDE ha sido utilizada por otros dispositivos -como por ejemplo, el nodeMCU (<http://nodemcu.com>), basado en el chip ESP8622, y adicionalmente, otras tarjetas de desarrollo para hardware libre han sido lanzadas al mercado, en la mayoría de los casos adaptando físicamente la tarjeta original a nuevas aplicaciones, y en otras integrando otros dispositivos digitales con más recursos que el ATmega328, tales como el ATmega1280, ATmega2560 e incluso dispositivos del tipo Atmel SAM3X8E ARM Cortex-M3, que permite trabajar en 32bits en forma nativa. El uso de un software de desarrollo

O.E. Amestica, P. E. Melin, C.R. Duran and G. Lagos are with the Dept. of Electrical and Electronic Engineering Universidad del Bio-Bio. Collao 1202, Concepción, Chile. e-mail: pemelin@ubiobio.cl

TABLA I DISPOSITIVOS ESTUDIADOS

Dispositivo	Procesador	Bit	Reloj MHz	Memoria (kB)			Digital I/O	Analog Input	Analog Output	Timer
				EEPROM	SRAM	Flash				
 Arduino UNO	Atmega328P-PU	8	16	1	2	32	14 (6 out PWM)	6 canales, 1 CAD de 10bit	0	2 de 8bits - 1 de 16bits
 Arduino MEGA	Atmega2560	8	16	4	8	256	54 (14 out PWM)	6 canales, 1 CAD de 10bit	0	2 de 8bits - 4 de 16bits
 Arduino DUE	AT91SAM3X8E	32	84	-	96	512	54 (12 out PWM)	16 canales, 1 CAD de 12bit	2	9 de 32 bits
 ESP8622	Xtensa LX106	32	80	-	-	512	10	1/10bit	0	1 por hardware
 ESP32	Xtensa LX6	32	240	-	520	520	32 (16 out PWM)	18 canales, 2 CAD de 12bit	2	4 de 84 bits

común, con instrucciones comunes independiente del dispositivo como lo es Arduino IDE permite que un usuario pueda migrar sus desarrollos de un dispositivo a otro de una forma tan simple como lo es definir el tipo de tarjeta de desarrollo a programar por Arduino IDE. Sin embargo, las diferencias entre cada equipo debido al microcontrolador de cada tarjeta generan las siguientes interrogantes con respecto a las capacidades de cada equipo, específicamente (i) para un mismo código escrito en Arduino IDE, ¿cuál es la diferencia entre los tiempos de ejecución entre una tarjeta de desarrollo y otra? ,o más importante y previo a la adquisición y desarrollo de un proyecto en una tarjeta de desarrollo, ¿Cumple el dispositivo seleccionado con las capacidades de procesamiento para realizar las acciones requeridas?.

Este trabajo presenta una comparación de tiempos de ejecución de acciones de dispositivos embebidos que pueden ser programados en Arduino IDE, con el objetivo de poder entregar un punto de referencia a desarrolladores con respecto a que plataforma de desarrollo pudiera cumplir con los requerimientos de su proyecto, o permitir evaluar a investigadores los tiempos de ejecución de nuevas estrategias de control automático. Para lo anterior, cinco placas de desarrollo típicamente usados para prototipaje rápido en aplicaciones relacionados con el control electrónico, son comparadas en términos del tiempo de ejecución de las instrucciones para (i) escritura de puertas digitales, (ii) adquisición de señales analógicas, (iii) operaciones matemáticas básicas, y (iv) ejecución del código de un controlador PI discreto, los cuales son requeridos para cualquier aplicación de control digital aplicado. La metodología para medir los tiempos de cada acción utiliza las puertas digitales de cada dispositivo, con el objetivo de poder medir el tiempo de ejecución utilizando un osciloscopio,

permitiendo replicar fácilmente la metodología con otros dispositivos embebidos.

II. PLATAFORMAS A EVALUAR

Los dispositivos a evaluar (ver Tabla I) se pueden dividir en dos categorías (i) equipos de hardware libre Arduino y (ii) equipos ESP que pueden ser utilizado aplicaciones Internet of Thing (IoT) debido a que integran hardware y antenas para su conexión a redes WIFI. Ambos tipos de equipos tendrán en común el hecho que puedan ser programados mediante la plataforma de desarrollo Arduino IDE (<https://www.arduino.cc>).

Dentro de los equipos de Hardware Libre Arduino se estudiarán Arduino UNO, Arduino MEGA y Arduino DUE. Todos son simplemente conocidos como Arduinos, pero dentro de la gama ya existen diferencias importantes, como los 8 bits de los procesadores de Arduino UNO y MEGA, contra los 32 bits de Arduino DUE. En los tres casos se mantiene la disposición física de pines en la tarjeta, lo que permite utilizar tarjetas de expansión – denominados Shields – y expandir las capacidades de la tarjeta hacia aplicaciones tan diversas como el control de motores DC, la conexión a tarjetas SD y micro-SD, etc., facilitando el prototipaje rápido de equipos mas complejos.

Por otro lado, para equipos IoT se evaluarán los denominados NODEMCU, basados en el chip ESPx, en ambos casos de 32-bits, el que en los últimos años se ha vuelto un dispositivo popular para el desarrollo de dispositivos IoT, principalmente debido al desarrollo de librerías para ser programado en la plataforma Arduino IDE. Estos dispositivos integran un procesador Xtensa junto al hardware necesario para la comunicación inalámbrica como WIFI o Bluetooth. Estas tarjetas tienen compatibilidad con Arduino IDE pero su

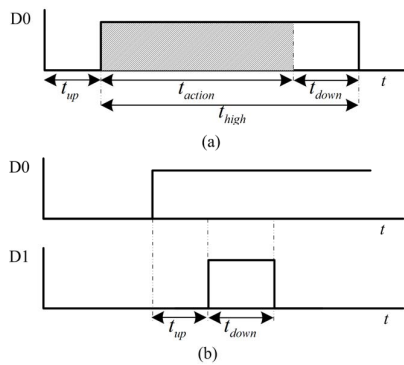


Fig. 1 Metodología de medición de acciones medición de los tiempos de ejecución (a) medición del tiempo de acción, (b) medición de tiempos de subida y bajada de puertos digitales

disposición de pines no es compatible con los Shield de Arduino.

A. Medición de las Acciones en cada Plataforma

La forma de evaluar tiempos de ejecución de acciones en un sistema microcomputarizado es medir el tiempo que se demora en ejecutar cada instrucción en lenguaje ASM,

```

1. int a= 1 ,b= 5 ,c2= 0,c3=0,c4=0, c5=0;
2. int c6=0, c7=0;
3. float d= 5.;
4.
5. void loop() {
6.   digitalWrite(D0, HIGH);
7.   digitalWrite(D1, HIGH);
8.   digitalWrite(D1, LOW);
9.   digitalWrite(D0, LOW);
10.
11.   d= d + 0.01;
12.   a= a+1;
13.   b= a+b;
14.
15.   digitalWrite(D2, HIGH);
16.   c2= a + b;
17.   digitalWrite(D2 ,LOW);
18.
19.   digitalWrite(D3, HIGH);
20.   c3= a - b;
21.   digitalWrite(D3 ,LOW);
22.
23.   digitalWrite(D4, HIGH);
24.   c4= a*b;
25.   digitalWrite(D4 ,LOW);
26.
27.   digitalWrite(D5, HIGH);
28.   c5= a/b;
29.   digitalWrite(D5 ,LOW);
30.
31.   digitalWrite(D6, HIGH);
32.   c6= int (d);
33.   digitalWrite(D6,LOW);
34.
35.   digitalWrite(D7, HIGH);
36.   c7=analogRead(A0);
37.   digitalWrite(D7 ,LOW);
38.
39.   delay(1);
40. }
```

Code 2 – Batch 1

utilizando la documentación del fabricante y la velocidad del reloj del equipo. Sin embargo, debido a la complejidad de las funciones implementadas típicamente por los usuarios, y a las facilidades entregadas por los lenguajes de alto nivel, es complejo acceder a las instrucciones en ASM y evaluar, una a una, su tiempo de ejecución. Por lo anterior, se considerará para este trabajo el uso de los puertos digitales para generar señales eléctricas que puedan ser medidas por un osciloscopio o un analizador lógico. Por lo mismo, en este trabajo se referirán los procesos a evaluar como acciones y no como instrucciones y/o comandos, para mantener la diferencia entre lo que pudiera hacer el compilador de alto nivel con respecto a lo que hace una instrucción en ASM. Por supuesto, debido a que cada compilador de alto nivel genera el código ASM para cada máquina, es posible que este pueda ser optimizado en el futuro, pudiendo el tiempo de las acciones medidas en este documento cambiar.

Para determinar el tiempo de ejecución de cada acción se utilizarán las salidas digitales de cada dispositivo, las que cambiarán de estado LOW a HIGH al comienzo de la acción, se mantendrá en nivel alto durante la acción y se cambiarán de HIGH a LOW al finalizar la acción. Lo anterior permitirá generar una señal en nivel alto que puede ser medida tal y como se muestra en Fig. 1a, y que queda definida por:

```

1. float a= 0.1 ,b= 5. ,c2= 0., c3=0., c4=0.;
2. float c5=0., c6=0., c7=0.;
3. int d= 5;
4.
5. void loop() {
6.   digitalWrite(D0, HIGH);
7.   digitalWrite(D1, HIGH);
8.   digitalWrite(D1, LOW);
9.   digitalWrite(D0, LOW);
10.
11.   d= d + 1;
12.   a= a+1.;
13.   b= a+b;
14.
15.   digitalWrite(D2, HIGH);
16.   c2= a + b;
17.   digitalWrite(D2 ,LOW);
18.
19.   digitalWrite(D3, HIGH);
20.   c3= a - b;
21.   digitalWrite(D3 ,LOW);
22.
23.   digitalWrite(D4, HIGH);
24.   c4= a*b;
25.   digitalWrite(D4 ,LOW);
26.
27.   digitalWrite(D5, HIGH);
28.   c5= a/b;
29.   digitalWrite(D5 ,LOW);
30.
31.   digitalWrite(D6, HIGH);
32.   c6= float(d);
33.   digitalWrite(D6 ,LOW);
34.
35.   digitalWrite(D7, HIGH);
36.   c7=cPI( a, b); // Ver Code 3
37.   digitalWrite(D7 ,LOW);
38.
39.   delay(1);
40. }
```

Code 1 – Batch 2

TABLA II RESULTADOS EXPERIMENTALES

Dispositivo	digitalWrite(Puerta, Valor)		int a,b,c; a=a+1; b=a+b;						float a,b,c; a=a+1; b=a+b;					
	HIGH to LOW (t_{down})	LOW to HIGH	c=a+b; (μs)	c=a-b; (μs)	c=a*b; (μs)	c=a/b; (μs)	float(a); (μs)	CAD (μs)	c=a+b; (μs)	c=a-b; (μs)	c=a*b; (μs)	c=a/b; (μs)	int(a) (μs)	PI (μs)
Arduino UNO	3.69	3.55	0.427	1.249	0.877	15.93	6.62	114.91	10.872	12.222	8.502	31.402	4.687	48.152
Arduino MEGA	3.95	3.85	2.66	2.52	2.835	17.855	14.705	109.745	12.661	12.341	10.661	33.411	5.625	49.411
Arduino DUE	2.07	2.08	0.15	0.21	0.19	0.17	0.59	4.26	1.31	1.29	1.02	2.61	0.74	7.23
ESP8622	0.915	0.645	0.125	0.125	0.16	0.44	0.5	95.085	0.69	0.73	1.225	3.415	0.5	5.99
ESP32	0.1225	0.098	0.0275	0.015	0.0275	0.0275	0.015	9.4775	0.025	0.0375	0.025	0.2125	0.025	0.2375

```

1. float I,y, P;
2. float cPI( float Ref, float Medida){
3. float kp = 3, Ki= 0.33, T=0.033, e;
4. e = Ref - Medida; // error
5. I= Ki*e*T; // parte integrativa
6. P=kp*e; // parte proporcional
7. y=y+ I+P;
8. return(y);
9. }

```

Code 3 – PI discretizado utilizando aproximación Euler.

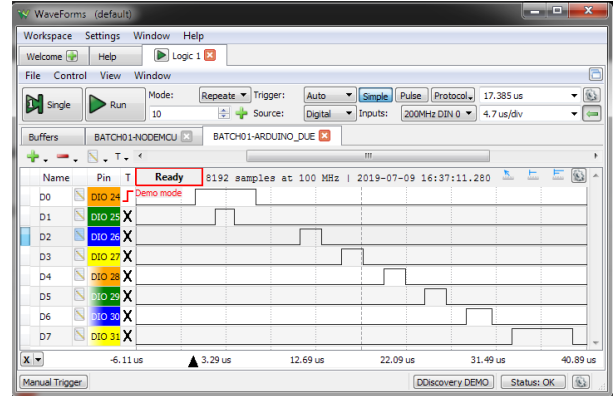


Fig. 2 Ejemplo de señales obtenidas con Batch 1, en Arduino DUE, utilizando el software WaveForms™

$$t_{high} = t_{action} + t_{down} \quad (1)$$

y donde t_{down} es determinado previamente mediante el uso de dos señales digitales cambiando su nivel lógico alternadamente, como se muestra en la Fig. 1b. Luego, el tiempo de cada acción queda definido por:

$$t_{action} = t_{high} - t_{down} \quad (2)$$

B. Set de Acciones a Evaluar

Las acciones a evaluar corresponden a acciones básicas y requeridas para el uso de un sistema digital en plataformas de monitoreo y control, es decir: adquisición de entradas y escritura de salidas, uso de operaciones aritméticas, acceso a memoria para escritura y lectura de datos, y la ejecución de un algoritmo de procesamiento de señales. Para esto se definieron dos batch de funciones (ver Code 2 y Code 1) de forma de ejecutar secuencialmente las acciones para poder visualizarlas mediante un osciloscopio o un analizador de funciones. Específicamente, las acciones a evaluar en este trabajo son:

- Salidas Digitales, específicamente el costo en tiempo de cambiar el estado de HIGH a LOW y de LOW a HIGH. Se descarta el uso de conversión digital analógica debido a que no todos los dispositivos evaluados lo incorporan dentro de sus hardware.
- Conversión analógica-digital, la cual se realizará utilizando un solo canal, con la máxima resolución y velocidad de reloj que permita el dispositivo.

- Operaciones Aritméticas, en formato entero (int) y punto flotante (float). Este último formato no está presente en forma de hardware en todos los dispositivos, pero se incluye en la evaluación debido a que los usuarios típicamente usan este formato gracias al compilador.
- Controlador PI, discretizado utilizando aproximación por Euler (ver Code 3).

Para todas las acciones se utilizarán los comandos disponibles por la plataforma de desarrollo. Así y por ejemplo, en el caso del cambio de estados de las puertas digitales, se utilizará el comando digitalWrite (Puerta, Valor) para la adquisición analógica digital se utilizará el comando analogRead (Puerta). En el caso de las operaciones matemáticas se utiliza la forma $c=a(OP)b$; , lo que considera dentro de la operación el costo de la llamada de las variables a operar (a y b), la asignación a una tercera variable (c).

C. Equipo Utilizado para la Medición del Tiempo de Ejecución

Para la medición del tiempo se utilizará el analizador lógico Digital Discovery, fabricado por Digilent. El equipo cuenta con 32 entradas/salidas digitales con niveles de 1 a 3.3V CMOS, 8 canales con una velocidad de muestreo de 800MS/s (usando puntas de prueba de alta velocidad), 16 canales con una velocidad de muestreo de 400MS/s, y 32 canales con una velocidad de muestreo de 200MS/s. Para este trabajo, se utilizará el equipo configurado en 200MS/s, utilizando conectores de alta velocidad suministrados por Digilent. Utilizando los códigos presentados en Code 2 y Code 1, y 8

entradas digitales del analizador, se obtienen formas de onda como la presentada en la Fig. 2, la que permite medir los tiempos de ejecución de cada acción referidas a las instrucciones de los códigos de prueba. La metodología propuesta permite que pueda ser replicado utilizando osciloscopios con canales analógicos o analizadores lógicos indistintamente.

III. RESULTADOS

A. Compatibilidad con Arduino IDE

Los cinco dispositivos testeados tienen compatibilidad con Arduino IDE. En el caso de los dispositivos Arduino UNO y MEGA, estas vienen preinstaladas en la plataforma en el momento de la instalación, mientras Arduino DUE debe ser instalado desde el Gestor de Tarjetas, desde las fuentes incluidas en Arduino IDE.

En el caso de las placas basadas en ESPx, estas deben ser instaladas desde las fuentes de sus fabricantes. Para esto y en las Preferencias de Arduino IDE se debe agregar al Gestor de URLs de Tarjetas Adicionales:

```
https://dl.espressif.com/dl/package_esp32_index.json
http://arduino.ESP8622.com/stable/package_ESP8622com_index.json
```

y a continuación agregar las tarjetas ESPx desde el Gestor de Tarjetas de Arduino IDE. Una vez realizado lo anterior, es necesario definir los puertos de salida para cada D_x definido en los Batch utilizados. Con esto se obtiene la compatibilidad con los otros comandos utilizados en los Batch (`analogRead`, `digitalWrite`, etc). Los resultados por operación son presentados en Tabla II - donde en cada función se destaca en rojo el dispositivo más lento y en verde el dispositivo más rápido -y son discutidos a continuación.

B. Escritura de Puertas Digitales

Las tres tarjetas Arduino tienen tiempos de escritura de las puertas digitales sobre 2 μ s, y diferencias entre los tiempos de escritura de HIGH to LOW y LOW to HIGH. Específicamente en dispositivos Arduino, el dispositivo más rápido es Arduino DUE (2.07 μ s) y el más lento es Arduino MEGA (3.95 μ s), solo 260 ns más lento que Arduino UNO (3.69 μ s). Respecto a LOW to HIGH, los tiempos de escritura son cerca de 100 ns más rápidos que de HIGH to LOW.

Es importante mencionar que para la escritura de la puerta digital se utilizó el comando `digitalWrite(Puerta, Valor)`. En todos los casos revisados, la escritura se realiza de forma más rápida al declarar el puerto de salida y escribir el bit o el puerto con el valor requerido.

C. Operaciones Matemáticas en Formato Entero

Arduino IDE define el formato `int` utilizando un registro de 16 bits. De los resultados obtenidos, el dispositivo que presenta el menor desempeño es Arduino MEGA, con tiempos de ejecución desde 2.66 μ s para la suma hasta 17.855 μ s para la división, mientras que el más rápido es ESP32, con tiempos de procesamiento menores a 100 ns. Considerando únicamente dispositivos Arduinos, el más rápido entre los tres es Arduino

DUE, con tiempos de ejecución entre 0.15 ns (suma) hasta 0.59 ns (conversión de `float` a `int`).

D. Operaciones Matemáticas en Formato Punto Flotante

De los dispositivos analizados solo ESP32 incorpora una unidad de punto flotante (FPU – Floating-Point Unit), por lo que al definir variables tipo flotante en Arduino IDE necesariamente el compilador deberá generar el código necesario para generar el formato numérico con los recursos de cada procesador. Es por esto que la diferencia entre los tiempos de ejecución de las instrucciones. Arduino UNO y Arduino MEGA, al ser utilizar de 8 bits, tienen tiempos que van desde los 10.87 μ s (suma en Arduino UNO) a tiempos que van hasta los 33.41 μ s (división en Arduino MEGA). Estos tiempos se reducen en el caso de Arduino DUE (1.31 μ s para la suma a 2.61 μ s para la división).

En el caso de los ESPx, los tiempos de ejecución son menores a los 3.5 μ s, existiendo una gran diferencia entre ESP8622 y ESP32. Específicamente, mientras los tiempos de ejecución de ESP8622 están entre 0.5 μ s (conversión de `int` a `float`) hasta 3.415 μ s (división), los tiempos en el ESP32 están bajo los 215 ns (división), y son menores a 30 ns en el caso de las otras operaciones. Esto debido al FPU disponible en el ESP32.

Con respecto a las conversiones de entero a flotante, se mantienen las diferencias, con tiempos de 5.6 μ s en el caso de Arduino UNO a 25 ns en el caso de ESP32.

E. Conversión Analógica Digital

La conversión analógica a digital es un punto clave en la adquisición de datos, siendo típicamente una operación lenta comparada con otras operaciones, y donde en dispositivos embebidos el CAD es multiplexado entre varios canales de entrada.

En los dispositivos evaluados, Arduino UNO, Arduino MEGA y ESP8622 tienen CAD de 10 bits, con la diferencia de que Arduino MEGA tiene 16 canales contra los 6 de Arduino UNO y solo un canal disponible en ESP8622. Estos dispositivos tienen tiempos de conversión cercanos los 100 μ s (95.085 μ s en ESP8622, 109.745 μ s en Arduino MEGA y 114.91 en Arduino UNO), mientras que los otros dispositivos tienen tiempos de conversión menores a 10 μ s. Específicamente, Arduino DUE (1 CAD de 12 bits multiplexado en 16 canales) tiene tiempos de conversión de 4.26 μ s, mientras que los ESP32 (18 canales y 2 CAD de 12 bits) tiene tiempos de conversión de 9.47 μ s. Cabe mencionar que los 18 canales podrían no estar disponibles en el pinout del dispositivo, mientras que para este trabajo no se consideró el uso de los dos CAD debido a que las modificaciones en el código no mantiene la compatibilidad de este en otras plataformas.

F. Evaluación del PI

La evaluación del PI como fue definido en este trabajo requiere el uso de formato numérico `float`. De los dispositivos estudiados, los más lentos fueron Arduino UNO y Arduino MEGA (48.152 μ s y 49.411 μ s respectivamente),

mientras que los dispositivos de 32 bits Arduino DUE fue el mas lento (5.99 μ s) y ESP32 el más rapido (237 ns). De los resultados se muestra que no es posible extrapolar directamente el costo de las operaciones por separado para determinar el tiempo de ejecución del PI.

IV. CONCLUSIONES

Este trabajo evalua y compara cinco tarjetas de desarrollo compatible con Arduino IDE, midiendo el tiempo de ejecución de instrucciones tipicamente usadas para aplicaciones de adquisición de datos y control digital. Las tarjetas corresponden a dispositivos Arduino UNO, Arduino MEGA y Arduino DUE, y a dispositivos ESPx (ESP8622 y ESP32), las cuales si bien no son tarjetas desarrolladas por el grupo Arduino, sus fabricantes proporcionan recursos para que su desarrollo sea compatible con la plataforma Arduino IDE. De los resultados obtenidos se observan pocas diferencias en los tiempos de ejecución de Arduino UNO y MEGA -ambos basados en procesadores de 8 bits-, pero si se observa un mejor desempeño por parte de la tarjeta Arduino DUE la que utiliza un reloj más rápido y utiliza un procesador de 32 bits. Por otro lado, en dispositivos ESPx, la tarjeta basada en ESP32 - procesador de 32 bits y que utiliza 240 MHz- supera ampliamente en capacidades de procesamiento tanto a la ESP8622 como a sus pares Arduino, teniendo unicamente diferencias en la velocidad de conversión analógica digital (9.47 μ s contra 4.26 μ s en Arduino DUE).

REFERENCES

- [1] Bangemann, S. Karnouskos, R. Camp, O. Carlsson, M. Riedl, S. McLeod, R. Harrison, A. W. Colombo, and P. Stluka, "State of the art in industrial automation," in *Industrial Cloud-Based Cyber-Physical Systems*, A. W. Colombo, T. Bangemann, S. Karnouskos, J. Delsing, P. Stluka, R. Harrison, F. Jammes, and J. L. Lastra, Eds. Springer, Cham, 2014, pp. 23–47.
- [2] S. W. A. Hashmi, M. Rehan, M. Aamir, H. Kumar, and F. Liaquat, "Distributed process monitoring and control using fpga," in *2014 4th International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace Electronic Systems (VITAE)*, May 2014, pp. 1–5.
- [3] S. Reddy, "Embedded dsp devices," in *Secure Smart Embedded Devices, Platforms and Applications*, K. Markantonakis and K. Maye, Eds. Springer New York, 2014, pp. 27–48.
- [4] A. Powell, "Democratizing production through open source knowledge: from open software to open hardware," *Media, Culture & Society*, vol. 34, no. 6, pp. 691–708, 2012. [Online]. Available: <https://doi.org/10.1177/0163443712449497>
- [5] C. Harnett, "Open source hardware for instrumentation and measurement," *IEEE Instrumentation Measurement Magazine*, vol. 14, no. 3, pp. 34–38, June 2011.
- [6] R. R. Santos, "Open hardware platforms in a first course of the computer engineering undergraduate program," in *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, Oct 2014, pp. 1–7.
- [7] L. M. Herger and M. Bodarky, "Engaging students with open source technologies and arduino," in *2015 IEEE Integrated STEM Education Conference*, March 2015, pp. 27–32.
- [8] A. Garrigos, D. Marroquí, J. M. Blanes, R. Gutiérrez, I. Blanquer, and M. Cantó, "Designing arduino electronic shields: Experiences from secondary and university courses," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, April 2017, pp. 934–937.
- [9] D. Costa and C. Duran-Faundez, "Open-Source Electronics Platforms as Enabling Technologies for Smart Cities: Recent Developments and Perspectives," *Electronics*, vol. 7, no. 12, p. 404, Dec. 2018.