Memorias del Congreso Nacional de Control Automático
ISSN: 2594-2492

Lenguaje para emular sistemas dinámicos en microcontroladores de 32 bits

Francisco Henández * **Domingo Cortés** * **Jorge Resa** *
Diego Martínez *

* *Instituto Politécnico Nacional, ESIME Culhuacán.*
(email: <fhernandezs094@gmail.com>, <domingo.cortes@gmail.com>,
<jorgeresa@yahoo.com>, <diegomtzg@hotmail.com >)

Resumen. Se desarrolla el software y procedimientos necesarios para hacer que un microcontrolador de 32 bits, pueda emular el comportamiento de un sistema dinámico descrito en un lenguaje de alto nivel. Para ello se desarrolla un lenguaje denominado Systdynam que pueda describir fácilmente un sistema dinámico y que a la vez sea fácil de traducir a código C; se desarrolla un traductor de Systdynam a código C; se desarrollan los procedimientos necesarios, para ejecutar ese código en tiempo real en un microcontrolador.

Keywords: Aplicaciones de control, Programación rápida, Sistemas embebidos, Desarrollo rápido de prototipos, Educación en control.

1. INTRODUCCIÓN

En este trabajo se presenta el desarrollo del software necesario para programar rápidamente un microcontrolador de 32 bits (MCU de 32-bits) con el fin de que pueda emular un sistema dinámico. Más precisamente, se desarrolla un sistema, para tener un dispositivo que pueda verse desde el exterior como una caja negra que acepta señales eléctricas como entradas y proporciona señales eléctricas como salidas y cuya dinámica sea programable desde un lenguaje de alto nivel.

Lo anterior se logra con tres componentes:

1. El diseño de un lenguaje de dominio específico [Baron (1986), van Deursen et al. (2000)]. Para la descripción de sistemas dinámicos.
2. La realización de un traductor de código [Clocksin (1997), Aho et al. (2011)] para que a partir de la descripción realizada en el punto 1 se genere código C que pueda ser compilado en un MCU de 32-bits .
3. El diseño de los procesos necesarios para que el código generado en el punto 2 se pueda ejecutar en el MCU de 32-bits bajo un sistema operativo de tiempo real.

Estos tres componentes permiten programar rápidamente un MCU de 32-bits para que pueda emular con características de tiempo real, una gran diversidad de sistemas dinámicos. Como un controlador es también un sistema dinámico, los mismos componentes permiten emular controladores.

Aunque sistemas similares existen en el mercado (Matlab, LabView, dSpace), estos generalmente tienen las siguientes desventajas:

1. Son de alto costo, lo que los hace frecuentemente inaccesibles.
2. Son cerrados, lo que limita su uso en la enseñanza y la experimentación.
3. Mantienen sus procesos ocultos por lo que no permiten construir nuevos, a partir de los ya existentes en sus respetivas plataformas.
4. Pueden llegar a dificultar la ejecución en tiempo real, por realizar los procesos propios del computador.

El esquema desarrollado permite ser aplicado para la enseñanza de diversas disciplinas como microcontroladores, programación, control, etc. Permite también el desarrollo rápido de controladores y prototipos, además, puede servir para hacer simulaciones con hardware en el lazo (HIL).

En la Sección 2, se describe el lenguaje de dominio específico denominado Systdynam, desarrollado para que sea sencillo describir sistemas dinámicos y a la vez sea fácilmente traducible. En la Sección 3 se describe cómo se construyó el traductor de Systdynam a código C. En la Sección 4 se describe cómo se hace que el código C generado por el traductor se ejecute en tiempo real. En la Sección 5 se muestran un par de ejemplos con los que se evaluó el software desarrollado y finalmente en la última sección se dan algunas conclusiones.


```
Parametros:
   ti:=0,
   tf:=10,
   dt:=0.01,
   entradas:=1,
   salidas:=2,
   scala_ent:=2,
   scala_sal:=2;
Sistema:
   #u,
   l=1,
   b=2
   g=9.8,
   m=3,
   j=m*l*l,
   x1'=x2,
   x2'=(1/j)*(-b*x2-m*g*l*sin(x1)+u),
   $y1=x1,
   $y2=x2;
Ciniciales:
   x1_0= 0.75,
   x2_0= 0;
```

Código 1. Descripción en Systdynam para la simulación de un péndulo simple.

2. SYSTDYNAM: UN LENGUAJE DE DOMINIO ESPECÍFICO PARA LA DESCRIPCIÓN DE SISTEMAS DINÁMICOS

En esta Sección se describe el lenguaje Systdynam: un lenguaje de dominio específico para la descripción de sistemas dinámicos y que a la vez es conveniente para realizar un traductor. En el Código 1 se ilustra la descripción de un péndulo simple en este lenguaje.

La descripción de un sistema en el lenguaje Systdynam, consta de tres bloques: parámetros, sistema y condiciones iniciales. Cada bloque empieza con una palabra clave y termina con ";". Las palabras claves de inicio de los bloques son:"Parametros:"; "Sistema:" y "Ciniciales:".

En el bloque que inicia con "Parametros:" se declaran aspectos del sistema que tienen que ver con el hardware. Los parámetros que se pueden especificar en este bloque están definidos en la Tabla 1. Al terminar cada parámetro se introduce ",".

En el bloque de "Sistema:" se describe propiamente el sistema dinámico mediante ecuaciones de estado. Los estados son identificados como los símbolos que preceden al "'". Así cuando se escribe $x2'$ se identifica al símbolo $x2$ como una variable de estado. En este bloque también, se pueden definir símbolos auxiliares como $b = 5$, de tal manera que el símbolo b, se pueda usar después. Los símbolos para las entradas deben estar precedidas de # y los símbolos de las salidas por $. Es necesario que todos los símbolos que no son estados, sean especificados antes de las ecuaciones de estado. Así en $x2' = b * x1$, es necesario haber especificado b anteriormente. Además para la especificación del sistema, se pueden utilizar cualquier función definida en la biblioteca de funciones matemáticas estándar en C, '$math.h$'.

Parámetro	Se usa para determinar:	Ejemplo
ti:	Tiempo de iniciar de la emulación.	ti: = 0
tf:	Tiempo final de la emulación.	tf: = 10
dt:	Paso de integración/periodo de muestreo.	dt: = 0.1
entradas:	Entradas del sistema.	entradas: = 1
salidas:	Salidas del sistema.	salidas: = 2
scala_ent:	Correspondencia a N unidades en el sistema, por cada un volt de entrada.	scala_ent: = 2.0
scala_sal:	Correspondencia a N unidad en el sistema, por cada volt de salida.	scala_sal: = 2.0

Tabla 1. Palabras reservadas de la sección "Parametros:".

Para especificar las condiciones iniciales en el bloque de "Ciniciales:", cada símbolo que representa un estado es sucedido por "_0". Así, si $x2$ representa a un estado, su condición inicial es determinada por $x2\_0 =< valor >$. Si existen mas de un estado inicial, cada condición es separada por una coma.

3. TRADUCTOR DE CÓDIGO DE SYSTDYNAM A LENGUAJE C

El lenguaje Systdynam, está diseñado de tal manera que sea sencillo describir un sistema dinámico, pero también para hacer sencilla la especificación léxica y sintáctica del lenguaje. De tal manera que un usuario con conocimientos en lenguaje C y un corto entrenamiento en analizadores léxico y sintácticos, pueda entender el proceso de traducción, de un sistema descrito en Systdynam a lenguaje C [Ellis and Stroustrup (1990)].

Para generar el traductor se usaron las herramientas de software libre FLex y Bison. FLex es una herramienta que a partir de un archivo, donde se establece la especificación léxica de un lenguaje, genera un analizador léxico para ese lenguaje.

La entrada de FLex es un archivo que contiene los patrones válidos del lenguaje, en este caso los patrones válidos de Systdynam. Esta especificación se hace principalmente por medio de expresiones regulares. Las expresiones regulares son un mecanismo para especificar patrones de texto, muy útiles para procesar texto, tanto que la mayoría de los editores de texto y lenguajes de programación incorporan bibliotecas para el uso de estas expresiones [Paxson et al. (2015)].

La salida de FLex, es un archivo en lenguaje C estándar, que al ser compilado, éste se vuelve un programa que es el analizador léxico o escáner del lenguaje Systdynam.


```
[()         { yylval.sym = yytext[0];   return OFNT; }
[)]         { yylval.sym = yytext[0];   return CFNT; }
[\-|\+]     { yylval.sym = yytext[0];   return OPA;  }
[\*|/]      { yylval.sym = yytext[0];   return OPA1; }
[,]         { yylval.sym = yytext[0];   return MORE; }
[=]         { yylval.sym = yytext[0];   return EQL;  }

[0-9]+[.]?[0-9]*     { strcpy(yylval.val, yytext);  return NUM;     }
[a-z][a-z0-9]*       { strcpy(yylval.val, yytext);  return VAR;     }
[a-z][a-z0-9]*[']    { strcpy(yylval.val, yytext);  return DEQQ;    }
[a-z][a-z0-9]*[_][0] { strcpy(yylval.val, yytext);  return INITALC; }


"Sistema:"      {  return STRT;     }
"Parametros:"   {  return STRPAR;   }
"Ciniciales:"   {  return STRINI;   }

"ti:"           {  return TINI;     }
"tf:"           {  return TFINAL;   }
"dt:"           {  return DTIME;    }
"entradas:"     {  return NINPUT;   }
"salidas:"      {  return NOUTPUT;  }
"escala_ent:"   {  return SCAL_IN;  }
"escala_sal:"   {  return SCAL_OU;  }
"#"             {  return IN_ID; }
"$"             {  return OUT_ID;}

";"             {  return STOP;     }

<<EOF>>     { return 0; }
[ \t\n]+    { }
.           { cout << "Token no encontrado!" << endl; exit(1); }
```

Figura 1. Tokens utilizados en el traductor de lenguaje Systdynam.

Este programa lee un archivo Systdynam con extensión 'txt' y lo descompone en 'tokens' del lenguaje. Que son las expresiones mínimas en el lenguaje. Con lo cual, determina cuáles símbolos representan el valor de los parámetros en la sección de "Parametros:"; cuales son los símbolos para los estados en la sección de "Sistema:", etc. Además FLex también permite especificar acción que se debe tomar cuando un token es encontrado. En la Figura 1 se muestra una parte de la especificación léxica de Systdynam.

Por otro lado Bison, es una herramienta que a partir de la especificación de la gramática de un lenguaje, genera el analizador sintáctico o "parser" de ese lenguaje. Donde la gramática se especifica en una notación propia de Bison [Donnelly and Stallman (2019)]. En este caso, la entrada de Bison es un archivo con la especificación de la gramática de Systdynam y la salida es el parser del lenguaje en código C. Que al ser compilado en conjunto con el archivo FLex, realizan el programa de aplicación, el cual es el compilador del lenguaje. Estos programas trabajan con conjunto, donde el parser solicita al analizador léxico cada uno de los tokens y ejecuta lad acciónes correspondientes. Generando así, el traductor de Systdynam a C.

Para dar una idea de como se expresan las reglas gramaticales, tomemos por ejemplo el conjunto de reglas que entran en juego para traducir el bloque "Sistema:". Éstas se muestran en la Figura 2. Para leer estar reglas es conveniente interpretar el carácter '|' como "o bien " y tener presente que, los tokens (palabras en mayúsculas) están definidas en la Figura 1. Sabiendo lo anterior, las reglas de la Figura 2 se pueden leer como sigue: El sistema se abstrae mediante un elemento llamado 'modelo'. Un 'modelo' es válido si tiene el token de inicio (STRT='Sistema:') seguido por una secuencia de símbolos llamada 'vvar' mas

```
modelo: STRT vvar STOP   {accion();}
vvar:   vvar MORE vvar   {accion();}
        | static         {accion();}
        | deq            {accion();}

static: VAR  EQL exp     {accion();}
deq:    DEQQ EQL exp     {accion();}

exp:    exp MORE exp             {accion();}
        | exp OPA1 exp           {accion();}
        | exp OPA0 exp           {accion();}
        | OPA0 exp %prec OPA1    {accion();}
        | OPA1 exp               {accion();}
        | ffun %prec VAR         {accion();}
        | NUMS                   {accion();}
        | VAR                    {accion();}
        | OFNT exp CFNT          {accion();}
 ffun:  VAR OFNT exp CFNT        {accion();}
```

Figura 2. Reglas gramaticales de la sección Sistema.

un token de fin (STOP=';'). La secuencia de símbolos 'vvar' está definida de forma recursiva. Una 'vvar' son dos 'vvar' separados por MORE (',') o bien una expresión 'static' o una expresión 'deq'. Una expresión 'static' es una VAR seguida de EQL (=) seguido de 'exp'. La cual se utiliza para definir variables estáticas. Y una 'deq' se utiliza para definir las variables de estado. Con esta idea, se puede interpretar el resto de la Figura 2.

Mediante la combinación de FLex y Bison, se genera el analizador léxico-sintáctico que traduce cada token del lenguaje en una acción semántica. Los detalles de cómo se generan las acciones semánticas a partir de un archivo en Systdynam son muy largos para describir aquí. Pero básicamente las operaciones matemáticas son traducidas a operaciones en el lenguaje C y las ecuaciones de estado son traducidas a una función que es integrada por el algoritmo de Euler. Se usa el algoritmo de Euler por facilidad y por predictibilidad en el tiempo de ejecución pero se podrían usar otros algoritmos de integración.

La gran parte de la traducción de Sysdynam ocurre durante el análisis léxico y sintáctico. Sin embargo, es necesario abordar algunos detalles en una sección de la especificación gramatical denominada "Bison Application File (BAF)".

La compilación en conjunto, del analizador léxico generado por FLex, el parser generado por Bison y el BAF, producen el código ejecutable que es el traductor de Systdynam. Con lo cual, el traductor acepta como entrada un archivo de texto que contienen la descripción de un sistema dinámico en Systdynam y devuelve una descripción del mismo sistema dinámico pero en código C.

Por motivos de espacio no se incluye aquí la especificación léxica completa de Systdynam, ni la especificación completa de su gramática; pero el lector interesado puede contactar a los autores para el código completo de ambas especificaciones.

4. EJECUCIÓN EN TIEMPO REAL DE UN MODELO DESCRITO EN SYSTDYNAM

Una vez que se tiene la descripción de un sistema dinámico en código C (obtenido a partir de una descripción en alto nivel), es posible compilarlo para hacer que se ejecute como la principal tarea de un sistema operativo de tiempo real (RTOS), corriendo en un sistema digital. Para ello, el sistema digital debe tener los siguientes requisitos mínimos.

- Ser capaz de correr un RTOS, tipo FreeRTOS, lo que por si solo implica la necesidad de una arquitectura de 32 bits, una cantidad mínima de memoria RAM y ROM, un sistema elaborado de interrupciones, etc.
- Tener la capacidad de hacer cálculos de punto flotante por hardware.
- Convertidores digital/analógico y analógico/digital.
- Software de desarrollo para implementar proyectos rápidamente y hacer depuración.

Hasta hace poco estos requisitos mínimos sólo se satisfacían con hardware relativamente costoso. Pero desde hace algunos años se introdujeron los MCUs 32-bits de bajo costo, que cuentan con unidades de punto flotante, memoria suficiente, y velocidad razonable para ejecutar RTOS. (Las figuras mostradas en el presente artículo, fueron obtenidas con periodo de muestreo de $10mS$, aunque se hicieron pruebas a $1mS$ sin dificultad, ambas a una frecuencia base de operación de 120MHz del MCU). Los MCUs que cuentan con lo anterior generalmente incluyen además, muchos periféricos como puertos ethernet, usb, SPI, I2C, UART, múltiples relojes, módulos PWM, PLL's, etc.

El proceso para configurar e instalar un RTOS y las tareas que se van a ejecutar, es altamente dependiente del sistema digital y requiere un conocimiento detallado del mismo. En el trabajo que se reporta se usó la tarjeta de desarrollo FRDM-K64F mostrada en la Figura 3 con el sistema operativo FreeRTOS. Dicha tarjeta tiene como núcleo un ARM Cortex-M4. Como los procesadores ARM se han vuelto tan comunes, existe mucho desarrollo para éstos. Esta tarjeta incluye el entorno de desarrollo integrado (IDE) MCUXpresso [Semiconductors (2018)]), que facilita el desarrollo de software, aunque por supuesto, no exime de conocer de manera detallada el funcionamiento de la misma. Esta IDE facilita mucho la configuración de la tarjeta, la instalación del FreeRTOS y la programación de las tareas para el sistema operativo.

FreeRTOS es un kernel o planificador de tareas en tiempo real para sistemas embebidos desarrollado y mantenido por el equipo de Real Time Engineers Ltd. Es un proyecto de código abierto de AWS y se distribuye bajo licencia MIT [Barry (2016)]. Este kernel permite que las aplicaciones se organicen como una colección de tareas independientes. Si se implementa en un procesador de un solo núcleo, el kernel decide por medio prioridades qué subproceso se debe ejecutar primero y cada prioridad es

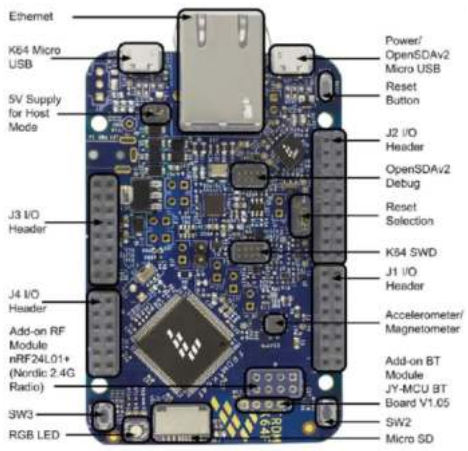

Figura 3. Tarjeta de desarrollo utilizada: FRDM-K64F.

asignada por el desarrollador de la aplicación. En este caso la única tarea es la ejecución del programa que emula el sistema dinámico.

5. EVALUACIÓN

Para evaluar el software y procedimientos desarrollados, se hizo la descripción en Systdynam de diversos sistemas dinámicos. Se procesó el código Systdynam por medio del traductor desarrollado para obtener código C y éste código se compiló para correrse en el MCU de 32-bits vía el sistema operativo FreeRTOS. Se tomaron mediciones de los pines de salida de la tarjeta con el osciloscopio y se compararon con una simulación realizada en Matlab-Simulink [Teng (2000)].

5.1 Evaluación con el modelo de un péndulo simple

A continuación se muestran los resultados para el péndulo simple del Código 1 de Systdynam.

Es importante hacer notar que el MCU de 32-bits utilizado (como la mayoría en la actualidad), está restringido a aceptar y sacar voltajes analógicos en el rango [0-3.3V]. Así no puede sacar los voltajes, que un ángulo negativo implicaría. Sería conveniente diseñar una tarjeta externa para sacar voltajes en un rango más amplio. Para este trabajo se decidió poner el cero en 1.65V, lo que representa una señal de offset. Con lo que se pueden mostrar valores negativos y positivos en el rango de [0,1.65),[1.65,3.3] volts. De la misma forma se realizó la simulación en Simulink teniendo las misma consideraciones para poder hacer la comparación.

Los resultados se muestran en las Figuras 4, 5 y 6. En la Figura 4 se muestra el resultado de la simulación en Simulink. En la figura 5 se muestran los datos numéricos que genera el proceso en el sistema digital, los cuales son posibles de ver gracias a las herramientas de depuración de la tarjeta. En la Figura 6 se muestra la señal analógica medida por un osciloscopio. Note la correspondencia entre las tres simulaciones.

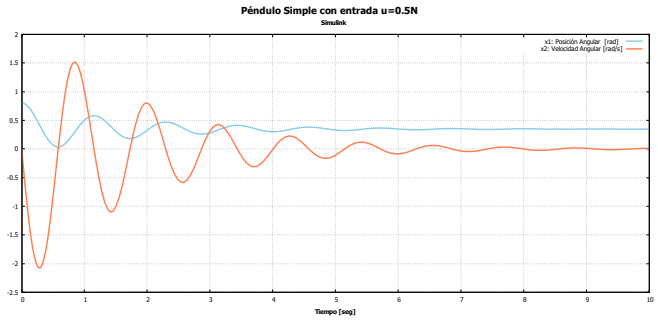

Figura 4. Simulación en Simulink del péndulo descrito en el Código 1.

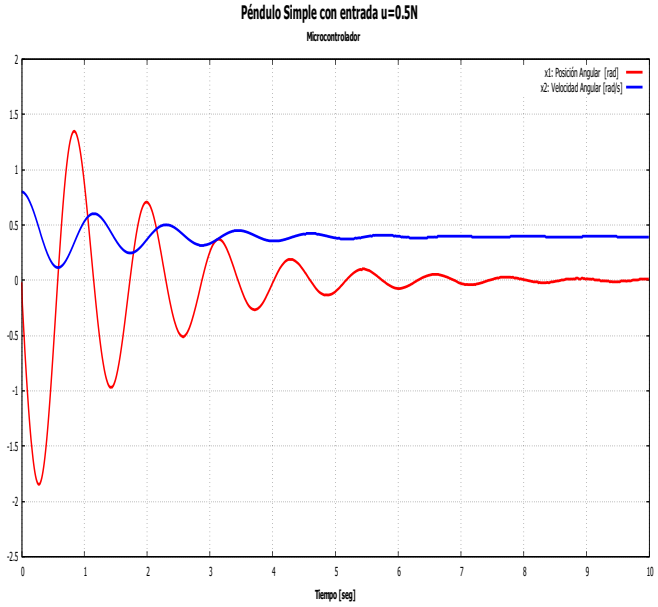

Figura 5. Valores generados por el MCU de 32-bits durante la simulación en tiempo real del péndulo descrito en el Código 1.

Es importante hacer notar que, si se quisiera que el sistema digital emulara otro sistema dinámico, la programación se podría hacer en cuestión de minutos, pues la descripción se hace desde un lenguaje de alto nivel.

5.2 Evaluación con el modelo de un oscilador de Van Der Pol

Considere el oscilador de Van Der Pol representado en ecuaciones de estado por

$$\dot{x}_1 = x_2 \tag{1a}$$

$$\dot{x}_2 = \xi(x_2 - \frac{x_2^3}{3}) - x_1 + u \tag{1b}$$

Para este caso en particular se fijó un coeficiente de amortiguamiento de: $\xi = 0.01$ y una señal de entrada constante igual a $u = 0$.

La descripción en Systdynam del modelo (1), se muestra en el Código 2. La señal de entrada que representa una constante con valor de 0 unidades, se colocó con una señal analógica constante igual a 2.475V. Debido al escalamiento de entrada y salida fijado en 2 y de la señal de offset de 1.65V.

Figura 6. Salida analógica del Código 1 simulado en el MCU de 32-bits en tiempo real.

```
Parametros:
    ti:=0,
    tf:=10,
    dt:=0.01,
    entradas:=1,
    salidas:=2,
    scala_ent:=2,
    scala_sal:=2,
Sistema:
    #u,
    epsi=0.01,
    x1'=x2,
    x2'=(epsi)*(x2-(x2*x2*x2)/3)-x1+u,
    $y1=x1,
    $y2=x2;
Ciniciales:
    x1_0= 0.75,
    x2_0= 0;
```

Código 2. Descripción en Systdynam del model (1).

En la Figura 7, se muestra de forma sobrepuesta, el resultado de simular el modelo (1) en simulink, y el resultado de la ejecución en tiempo real del código generado a partir de la descripción de la Figura 2. Los datos graficados en el caso del sistema digital, son los números generados en el programa. En dicha Figura se aprecia que la variación entre los resultados es muy pequeña.

En la Figura 8, se muestra la señal analógica medida en el osciloscopio. Note que la señal del osciloscopio está restringida al rango [0-3.3V] y ha pasado el efecto de truncamiento de la conversión digital/analógico. No obstante la señal obtenida es bastante fiel a la obtenida numéricamente en Simulink.

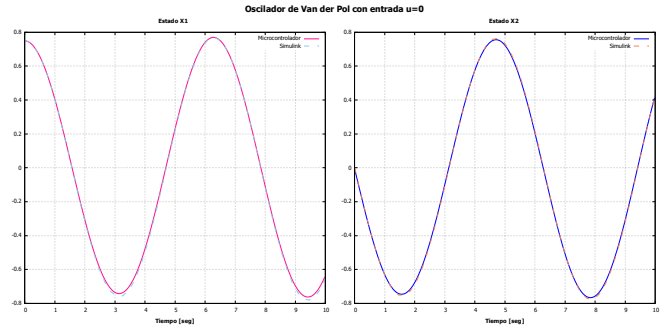

Figura 7. Comparación de los resultados de simulación del oscilador de Van Der Pol: Simulink vs Systdynam.

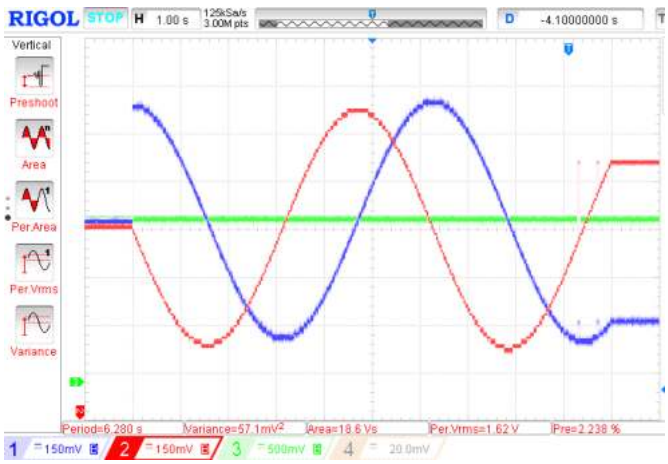

Figura 8. Señal analógica obtenida a partir de la descripción en Systdynam de un oscilador de Van Der Pol.

6. CONCLUSIONES

En este artículo se desarrolló un sistema de software que habilita a un MCU de 32-bits , comportarse como un sistema dinámico programable, con señales eléctricas como entradas y salidas. La descripción de dicho sistema dinámico se puede hacer desde un lenguaje de alto nivel.

Para lograr lo anterior se diseño el lenguaje Systdynam, se desarrollo el traductor de código Systdynam a código C y se desarrollaron los procedimientos necesarios para que una tarea expresada en código C se ejecute en tiempo real en un MCU de 32-bits .

El lenguaje Systdynam cumple con dos objetivos. Por una parte permite describir de manera sencilla un sistema dinámico a partir de las ecuaciones de estado. Por otra parte está diseñado de tal manera que su traducción sea relativamente fácil para que se pueda realizar sin necesidad de una capacitación exhaustiva sobre compiladores.

El traductor de Systdynam a código C, fue realizado utilizando el generador de analizadores léxicos: FLex y el generador de analizadores sintácticos: Bison. Con lo que se desarrollaron las especificaciones léxica y sintácticas del lenguaje Systdynam. Las cuales dan base a el traductor que toma como entrada una descripción de un sistema dinámico en Systdynam y devuelve la misma descripción en código C.

La descripción del sistema en código C se compila, junto con el código necesario para que el MCU de 32-bits lo ejecute como tarea principal mediante un RTOS. Estos procedimientos son altamente dependientes de la tarjeta elegida. Se determinaron los procedimientos para la tarjeta FRDM-K64F, que tiene cómo núcleo un ARM Cortex M4. En estos procedimientos fue de gran ayuda el IDE MCUXpresso y es imprescindible que la tarjeta pueda ejecutar un RTOS. La tarjeta utilizada en este trabajo puede ejecutar FreeRTOS.

Así, se tiene un sistema que puede programarse rápidamente para emular una gran diversidad de sistemas dinámicos. Dicho sistema pude usarse para hacer prototipos rápidamente, para hacer HIL y también para educación.

Con la ayuda de un MCU de 32-bits y el software desarrollado, se podrían apoyar los cursos de: programación, sistemas embebidos, sistemas en tiempo real, sistemas operativos, compiladores y control. Más aún, se puede desarrollar un curso de penúltimo o último semestre de ingeniería en computación, electrónica ó ramas afines para unificar una gran cantidad de conocimientos que los estudiantes ven aisladamente.

Todas las partes del sistema son perfectibles. El lenguaje se puede ampliar, el traductor se puede hacer mas eficiente y los procedimientos para ejecutarlo en el MCU de 32-bits, se pueden mejorar. Además se pueden generar una biblioteca de sistemas dinámicos y una biblioteca de controladores para la experimentación. Se pueden construir tarjetas externas para construir un hardware que acepte y entregue señales que cumplan algún estándar en la industria.

REFERENCIAS

Aho, A., Lam, M., Ullman, J., and Sethi, R. (2011). *Compilers: Principles, Techniques, and Tools*. Pearson Education.

Baron, N. (1986). *Computer Languages: A Guide for the Perplexed*. Anchor books. Anchor Press/Doubleday.

Barry, R. (2016). *Mastering the FreeRTOS Real TimeKernel A Hands On Tutorial Guide*.

Clocksin, W.F. (1997). *Clause and Effect: Prolog Programming for the Working Programmer*. Springer-Verlag, Berlin, Heidelberg.

Donnelly, C. and Stallman, R. (2019). *Bison Manual*, version 3.3 edition.

Ellis, M. and Stroustrup, B. (1990). *The Annotated C++ Reference Manual*. ANSI base document. Addison-Wesley.

Paxson, V., Estes, W., and Millaway, J. (2015). *Lexical Analysis with Flex*, 2.6.0 edition.

Semiconductors, N. (2018). *MCUXpresso IDE User Guide*. NXP Semiconductors.

Teng, F. (2000). Real-time control using matlab simulink: Windows and linux based systems. *IFAC Proceedings Volumes*, 199–204. doi:10.1016/S1474-6670(17)37863-1.

van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific Languages: An Annotated Bibliography. *SIGPLAN Not.*, 35(6), 26–36. doi:10.1145/352029.352035.

