Trabajo de Fin de Master

Síntesis automática de amplificadores electrónicos mediante *Grammatical Evolution*

Federico Castejón Lapeyra
Director: Enrique J. Carmona

Junio 2013

Universidad Nacional de Educación a Distancia
Escuela Técnica Superior de Ingeniería Informática

Agradecimientos

En primer lugar agradecer al director de este trabajo de fin de master, Enrique J. Carmona, por toda la dedicación, apoyo y confianza mostrados durante la realización del mismo.

Agradecer a Sara por todo el apoyo y comprensión, sin los que no habría podido hacer este trabajo, y especialmente a Susana por comprender que su padre también tenía que trabajar en casa.

Resumen

En este trabajo se presenta un algoritmo para la síntesis automática de amplificadores analógicos de una etapa basados en *Grammatical Evolution* (GE) y en las expresiones de desarrollo de Koza. GE es un algoritmo evolutivo capaz de generar código en cualquier lenguaje de programación, que utiliza cadenas de bytes de longitud variable. El proceso de decodificación de un cromosoma hace uso de una gramática BNF del lenguaje objetivo y se basa en recorrer la cadena de bytes, determinando cada uno de ellos qué regla de producción de la gramática será utilizada. Este proceso da lugar a una traducción de genotipo a fenotipo, generando una expresión perteneciente a la gramática utilizada.

En los trabajos de Koza y sus colaboradores se introduce una forma de representación de circuitos mediante árboles de parseado. Este tipo de representación, llamada de desarrollo, se basa en aplicar una sucesión de funciones de transformación sobre un circuito de partida, llamado embrión, y como resultado de este proceso de desarrollo, se obtiene un circuito final. El algoritmo desarrollado, basado en GE, utiliza una gramática compuesta basada en el conjunto de las funciones de transformación definidas por Koza. Hasta donde se sabe por la bibliografía relacionada con el área, la gramática aquí propuesta es la primera dedicada a abordar el diseño automático de circuitos analógicos mediante el uso de GE.

Se ha trabajado también en la creación de una función de adaptación compuesta por diferentes términos que permite hacer medible computacionalmente las especificaciones de diseño del amplificador objetivo. Toda esta metodología se ha aplicado a dos casos de estudio, mostrando y analizando los mejores circuitos obtenidos. Finalmente, dichos circuitos se comparan con los obtenidos por un diseñador humano y con los resultados obtenidos en otros trabajos relacionados con el diseño automático de amplificadores.

Índice general

1. Introducción

El presente trabajo se enmarca dentro del campo de la electrónica evolutiva, que busca la síntesis automática de circuitos electrónicos mediante el uso de algoritmos evolutivos. En dicho campo no sólo se está interesado en soluciones que cumplan los requisitos de diseño, sino además en que sean óptimas en términos de número de componentes, velocidad o consumo. Otra característica de la electrónica evolutiva es que debe cubrir las dos tareas de la síntesis de circuitos electrónicos: la selección de la topología y el dimensionamiento de los componentes.

La elección de *Grammatical Evolution*, entre los diferentes paradigmas evolutivos, se ha debido a las interesantes ventajas que presenta tanto en codificación, utilizando cadenas binarias, como en el uso de operadores de variación estándar. El modelo de representación de circuitos parte del trabajo de Koza (Koza et al., 1998; Koza et al., 1999b; Koza et al., 1999a; Koza et al., 2000a; Koza et al., 2000b), en el que define un proceso de desarrollo de los circuitos basado en expresiones que utilizan funciones de transformación sobre un embrión de circuito. A diferencia de Koza que utilizaba Programación Evolutiva, como se ha indicado anteriormente, este trabajo se centra en el uso de *Grammatical Evolution*.

Hasta donde se sabe por la bibliografía relacionada con el área, la gramática aquí propuesta es la primera dedicada a abordar el diseño automático de circuitos analógicos mediante el uso de GE.

El circuito objetivo ha sido un amplificador basado en transistores BJT. La elección de este objetivo ha sido debida a que ya existen trabajos en los que se ha estudiado la síntesis automática de filtros, mientras que hay menos estudios en el campo de los amplificadores. Por otro lado, también existen métodos analíticos de síntesis de filtros analógicos.

Se ha desarrollado un programa para implementar el algoritmo propuesto. Este programa se ha desarrollado en *Java* y utiliza *JavaCC* para la implementación del *parser* de las expresiones de desarrollo. El programa se ha paralelizado para su ejecución en un cluster de cinco nodos, lo que ha permitido reducir el tiempo de ejecución. La paralelización ha atendido a la evaluación de los circuitos que es la parte más costosa computacionalmente del algoritmo. Para la evaluación de los circuitos se ha utilizado *Ngspice* (Nenzi and Vogt, 2011), que es una evolución del conocido simulador de circuitos *SPICE3* (Nagel and Pederson, 1973).

Toda esta metodología se ha aplicado a dos casos de estudio, basados en el diseño de dos amplificadores con diferentes especificaciones. Se ha realizado la comparación de

los mejores circuitos obtenidos frente a un diseño realizado por un diseñador humano, y también, de una manera cualitativa, con los resultados obtenidos en otros trabajos relacionados con el diseño automatico de amplificadores.

La revisión de la bibliografía se realiza en el capítulo 2. El capítulo 3 describe el paradigma Grammatical Evolution, incluyendo un ejemplo de gramática y de decodificación de un cromosoma. El capítulo 4 explica en detalle la solución propuesta en este trabajo para el diseño automático de amplificadores, comenzando por el proceso de desarrollo y mostrando también la función de adaptación basada en medidas sobre el circuito. El capítulo 5 describe el programa realizado, incluyendo la forma de paralelización del algoritmo. El capítulo 6 contiene la evaluación del sistema propuesto, basada en cuatro experimentos de los que se muestran los dos mejores circuitos. El capítulo 7 contiene la discusión de resultados, incluyendo la comparación de los mejores circuitos frente a los diseñados por un diseñador humano. El último capítulo contiene las conclusiones.

2. Revisión de la Bibliografía

En este capítulo se realiza una revisión de la bibliografía del área del diseño de circuitos electrónicos analógicos y digitales, indicando las diferencias principales en el diseño de ambos. Posteriormente se aborda el campo de la electrónica evolutiva, que utiliza algoritmos evolutivos para la síntesis automática de circuitos electrónicos, y se realiza una descripción de los trabajos previos principales en este campo. Finalmente se describe la aproximación de Koza a la síntesis de circuitos electrónicos analógicos, indicando la importancia principal que tendrá para este trabajo.

2.1. Diferencias en el diseño de circuitos analógicos y digitales

La introducción del circuito integrado (IC, del inglés *integrated circuit*) en 1958 y posteriormente de los circuitos digitales, junto con su mayor sencillez para su síntesis, ha llevado a una digitalización en muchos campos donde antes sólo existía la alternativa analógica, siendo, por ejemplo, muy habitual el uso de procesadores digitales de señal (DSP, del inglés digital signal processor) en lugar de filtros analógicos.

Con el avance del dominio digital, se ha predicho el abandono de la tecnología analógica de forma continuada, si bien también hay opiniones contrarias(Camenzind, 2005; Gielen and Rutenbar, 2000; Rutenbar, 1993; Soderstrand, 2012). Ello es debido a que no todas las funcionalidades pueden ser digitalizadas, debiendo permanecer analógicas. Un ejemplo son los subsistemas de entrada y salida, pues los transductores son también analógicos: como sensores, micrófonos o antenas en el subsistema de entrada; y como altavoces, pantallas o motores en el subsistema de salida. Otra función analógica son las comunicaciones RF. En este sentido se ha indicado también que indica que "el mundo es fundamentalmente analógico" y que "la revolución digital está construida sobre una realidad analógica" (Camenzind, 2005).

Por otro lado, en altas frecuencias, los circuitos digitales presentan un comportamiento analógico (Tlelo-Cuautle et al., 2010), llegando a ser considerados como "los mayores circuitos analógicos hoy en día" (Gielen and Rutenbar, 2000).

Si atendemos al campo del diseño asistido por ordenador (CAD, del inglés *computer aided design*), que en el caso del diseño de circuitos electrónicos se convierte en el Diseño Electrónico Automatizado (EDA, del inglés *electronic design automation*), nos encontramos con que los circuitos electrónicos digitales disponen de herramientas

CAD o EDA más desarrolladas, en las que muchos aspectos están automatizados de forma completa. Los circuitos digitales pueden representarse de forma formal, existiendo los llamados lenguajes descriptivos del hardware (HDL, del inglés *hardware description language*), como son VHDL o Verilog (Gielen and Rutenbar, 2000).

En el caso de los circuitos electrónicos analógicos no existe una herramienta o metodología general para su diseño. En el campo EDA, se han llevado a cabo importantes avances para obtener herramientas que asistan al diseñador en esta tarea, tanto en el caso de circuitos integrados analógicos o de señal mixta, como por ejemplo la disponibilidad de librerías de celdas o el uso de sistemas expertos basados en reglas.

Esta diferencia entre circuitos analógicos y digitales da lugar a paradojas, como el caso de circuitos de señal mixta, en los que se consume más tiempo en el diseño de la parte analógica, que además es proporcionalmente más pequeña que en la parte digital, llegando a ser el cuello de botella(Gielen and Rutenbar, 2000).

Un ejemplo más de la diferencia entre circuitos analógicos y digitales es la disponibilidad de librerías de celdas estándar, donde las celdas son bloques de circuito preconstruidos que realizan una función individual y que permiten la construcción de circuitos mayores. En el caso digital, las librerías existentes cubren toda la funcionalidad y su agregación es directa. Sin embargo en el caso analógico no ocurre así. Por un lado no suelen estar disponibles todas las celdas necesarias, pues por ejemplo, en el caso de diseño de un ASIC suelen encontrarse entre un 70 % y un 90 % de las celdas disponibles, por lo que siempre es necesario diseñar algunas celdas a medida (Rutenbar, 1993). Por otro lado, también suele ser necesario modificar a medida las existentes (Antao, 1996). Para resaltar esta necesidad de modificación a medida de las celdas existentes, se ha llegado a decir que en el ámbito analógico “no existen celdas estándar”(Camenzind, 2005).

Con todo ello, los circuitos electrónicos analógicos todavía requieren ser diseñados por expertos en el campo, dando lugar a lo que se conoce como el dilema analógico (Aaserud and Nielsen, 1995), donde por un lado, el caso analógico es menos sistemático y también intensivo en conocimiento (Harjani et al., 1987; Gielen and Rutenbar, 2000), y por otro lado, existe una escasez debida a la predominancia de lo digital (Camenzind, 2005).

2.2. Fases de diseño de los circuitos electrónicos

El diseño de circuitos consta de dos tareas: la selección de la topología del circuito y la asignación de los valores de los parámetros que definen cada componente, recibiendo esta última tarea el nombre de dimensionamiento de los componentes. Ambas tareas deben dar lugar a un circuito que cumpla la funcionalidad requerida. En el caso de los circuitos integrados, además, es necesaria una tarea adicional para determinar la disposición espacial o *layout* del mismo, que busca disponer el circuito de una forma eficiente para su integración en un chip.

Como se ha comentado previamente, la síntesis de un circuito se puede descomponer en dos tareas: la selección de la topología y el dimensionamiento de los componentes. Estas tareas se pueden hacer de forma completamente separada o en paralelo (Martens and Gielen, 2008):

- Separación completa de las tareas pudiendo realizar primero la selección de la topología y posteriormente el dimensionamiento. También es posible dimensionar varias topologías candidatas y realizar la selección posteriormente al dimensionamiento.
- Selección y dimensionamiento en paralelo, existiendo varias opciones de topología, bien para la arquitectura completa o bien para subbloques, y permitiendo la combinación de dichas opciones y la selección entre ellas a partir del dimensionamiento.

La tarea de selección de la topología se puede hacer directamente por un diseñador humano basándose en sus conocimientos o experiencia, pero también las herramientas CAD pueden asistir en esta tarea mediante varios enfoques, como por ejemplo el uso de sistemas expertos basados en reglas. El dimensionamiento se realiza mediante algoritmos de optimización, como podrían ser *simulated annealing* o también los algoritmos evolutivos.

En este trabajo se atenderá al uso de algoritmos evolutivos no sólo para el dimensionamiento sino para la síntesis de la topología del circuito, entrando en lo que seria un enfoque ascendente o *bottom-up*, si atendemos a una descripción jerárquica de dicha tarea, frente a los enfoques más clásicos y por otro lado más utilizados que corresponderían a un enfoque descendente o *top-down*. Por tanto, la tercera fase, relacionada con el *layout*, no será abordada.

2.3. Electrónica evolutiva

La electrónica evolutiva surge en 1998 (Zebulum et al., 1998) y busca conseguir la síntesis automática de circuitos electrónicos mediante el uso de algoritmos evolutivos. Adicionalmente se buscan soluciones que no sólo cumplen los requisitos de diseño, sino que además son óptimas en términos de número de componentes, velocidad o consumo. Otra característica de la electrónica evolutiva es que debe cubrir las dos tareas de la síntesis de circuitos electrónicos: la selección de la topología y el dimensionamiento de los componentes. Hay trabajos que utilizan algoritmos evolutivos como una de las herramientas de optimización numérica para asistir a un diseñador humano en la tarea de dimensionamiento de un circuito, una vez la topología ya ha sido seleccionada (Puhan et al., 1999; Varios, 2013). Sin embargo, no se consideran dentro de la electrónica evolutiva.

Debido a la falta de métodos automatizados para la síntesis de los circuitos analógicos, hay un mayor interés en el campo de la electrónica analógica (Ando and Iba,

2000; Gan et al., 2010; Koza et al., 1999a; Mattiussi, 2005; Mühlenbein et al., 2007; Wang et al., 2007; Zebulum et al., 1998), si bien también hay trabajos en el campo de circuitos digitales (Coello Coello and Aguirre, 2002; Karpuzcu, 2005; Yan et al., 2006; Yan and Jin, 2010). En la literatura existen otros estudios bibliográficos del uso de algoritmos evolutivos en la síntesis de circuitos analógicos (Tlelo-Cuautle and Duarte-Villaseñor, 2008; Tlelo-Cuautle et al., 2010).

Los algoritmos evolutivos se inspiran en el proceso natural de la evolución y permiten obtener soluciones a problemas complejos mediante prueba y error, utilizando una población de soluciones tentativas que se reproducen de acuerdo al grado de adaptación. Las soluciones se recombinan y mutan siguiendo el símil natural, buscando obtener la supervivencia de los más aptos.

La síntesis mediante algoritmos evolutivos no sólo no utiliza reglas de diseño ni conocimiento experto en el diseño (Grimbleby, 2000), sino que permite encontrar soluciones no convencionales que desafían la intuición de diseñadores humanos (Eiben and Smith, 2008). Sin embargo, presenta la desventaja de que los circuitos obtenidos pueden no ser fácilmente analizables por diseñadores humanos o provocar rechazo en su aceptación.

La evaluación de la adaptación o fitness se realiza mediante un simulador de circuitos electrónicos, siendo *SPICE* el estándar de facto de la industria para circuitos analógicos. Este simulador fue desarrollado por Nagel y Pederson en la Universidad de California Berkeley (Nagel and Pederson, 1973). La distribución abierta del mismo contribuyó a su expansión, siendo también la base para desarrollos propietarios comerciales. La última versión de Berkeley es *SPICE3f4*.

La electrónica evolutiva está relacionada con el hardware evolutivo (EHW, del inglés *evolvable hardware*), en el que se utilizan algoritmos evolutivos para el diseño de circuitos implementados en una FPGA(Higuchi et al., 1996), lo que permite su programación y la evaluación de cada circuito en el propio hardware en lugar de realizar una simulación del mismo.

Un problema actualmente no cerrado en electrónica evolutiva es la representación de un circuito en un cromosoma, por lo que se han propuesto varias representaciones en la literatura, que se podrían clasificar, por ejemplo (Mattiussi and Floreano, 2007), como sigue:

1. Codificación directa. El cromosoma codifica directamente los componentes y sus conexiones, lo que resulta muy sencillo para su decodificación. Sin embargo, debido a que es necesario almacenar cada conexión, el cromosoma tiende a crecer mucho de tamaño con la complejidad del circuito. Finalmente también tiene la contrapartida de requerir unos operadores de cruce y mutación complejos para mantener la viabilidad del circuito.
2. Codificación de desarrollo. El cromosoma almacena la secuencia de transformaciones que se realizan a partir de un circuito embrión hasta hasta alcanzar un circuito final. A diferencia de la codificación directa, presenta un mejor comportamiento frente al aumento de complejidad del circuito.

3. Interacción implícita. Esta forma de codificación está inspirada en las redes de regulación genética (GRN) biológicas, en las que los genes disponen de zonas de regulación en la que reciben el mensaje de otros genes presentes en el cromosoma para regular su expresión.

La representación elegida también tiene un impacto en los operadores de cruce y mutación, pues deben garantizar la consistencia de los cromosomas obtenidos tras la aplicación de dichos operadores.

Finalmente, el estado actual de la electrónica evolutiva es el de desarrollos experimentales, para el diseño de circuitos de pequeño o mediano tamaño. Uno de los problemas que existen es el de la escalabilidad debido a la explosión combinatoria de las soluciones y los elevados tiempos de simulación de circuitos grandes o muy grandes. Otro de los problemas pendientes es el paso del diseño experimental a diseños industriales, pues en general, los diseños obtenidos no se han implementado en circuitos reales, habiendo sido su estudio en el campo de la simulación, o al menos no existe información sobre los resultados reales (Stoica et al., 2004).

2.3.1. Trabajos previos en síntesis de circuitos electrónicos analógicos

En esta sección se comentan trabajos previos realizados en electrónica evolutiva, prestando atención a la forma de representación utilizada. A modo de resumen, la Tabla 2.1 muestra los trabajos previos en el campo de la electrónica analógica.

Equipo	Representación	Tipo de codificación	Tipo de algoritmo evolutivo	Año
(Koza et al., 1999a)	Árboles de parseado	desarrollo	PG	1999
(Ando and Iba, 2000)	Lista de componentes y conexiones	directa	Messy GA	2000
(Grimbleby, 2000)	Lista de componentes y conexiones	directa	Hybrid GA	2000
(Zebulum et al., 2000)	Lista de componentes y conexiones	directa	SAGA	2000
(Mattiussi, 2005)	AGE	Interacción implícita	GRN	2005

Tabla 2.1. – Trabajos previos en electrónica analógica

Codificación directa

La representación almacena el conjunto de componentes que componen el circuito y las conexiones entre los mismos. En algunos casos se almacenan también los valores de los mismos para la evolución conjunta de la topología y los valores de los componentes. En otros casos, no se almacenan los valores, realizando la tarea de dimensionamiento mediante un optimizador numérico. Esta optimización puede hacerse de forma posterior a la determinación de la topología (Ando and Iba, 2000), o bien, incorporando la optimización dentro de la evaluación de cada individuo (Grimbleby, 1995; Grimbleby, 2000). El optimizador puede ser también un algoritmo evolutivo (Ando and Iba, 2000).

Con el fin de poder representar circuitos de diferentes tamaños, se suele utilizar codificación de longitud variable. Sin embargo, también se utiliza una codificación con un tamaño fijo máximo, y se permite la representación de circuitos más pequeños, lo que en realidad corresponde a una codificación de longitud variable con un limite máximo. En estos casos, la forma de permitir circuitos más pequeños se realiza bien mediante la posibilidad de componentes nulos (Grimbleby, 1995; Grimbleby, 2000), o bien mediante desactivación de genes, para lo que se usa una cadena de valores de activación de los genes como parte del cromosoma (Zebulum et al., 1998; Zebulum et al., 2000).

El algoritmo evolutivo utilizado suele ser un algoritmo genético, o bien variaciones del mismo, que además puede haberse modificado para acoplar el optimizador numérico. Las poblaciones utilizadas son bajas, entre 30 y 200 individuos, excepto en un caso que se usan poblaciones más grandes, entre 500 y 2000 (Ando and Iba, 2000).

Uno de los problemas de este tipo de representación es que los operadores de variación deben ser más complejos que en los algoritmos genéticos estándar, con el objetivo de mantener la viabilidad de los circuitos resultantes. El operador de cruce puede ser más parecido al estándar, si bien, suele operar a nivel de gen, donde gen se define como un componente y sus conexiones. Sin embargo, el operador de mutación debe ser diseñado ad hoc para el tipo de representación, de forma que se garantice la viabilidad del cromosoma mutado, o bien, que al menos se maximice el número de resultados viables. Por ello, se suele limitar el número de transformaciones posibles para que no sean muy destructivas. Por ejemplo, sustitución de un componente por un cortocircuito o circuito abierto, o bien, conexión de un nuevo componente en serie o paralelo sobre un componente existente (Grimbleby, 2000). Finalmente, en algún caso es necesario introducir un operador de variación, que en este caso es un operador de variación de longitud y que opera sobre la cadena de valores de activación de genes (Zebulum et al., 1998; Zebulum et al., 2000).

En los casos en los que se usa longitud variable se observa aparición del efecto engorde, o *bloat*, siendo necesario utilizar mecanismos para limitar el crecimiento de los cromosomas, llamados de parsimonia, y así obtener soluciones eficientes en el número de componentes de los circuitos sintetizados (Zebulum et al., 2000). Incluso

en los casos de longitud fija, que como se ha indicado previamente, son realmente representaciones de longitud variable con un límite máximo, se menciona la introducción de mecanismos de parsimonia (Grimbleby, 1995). El efecto engorde también tiene un impacto computacional, incrementando los tiempos de ejecución al considerar soluciones innecesariamente complejas. Esta es otra razón más para incorporar mecanismos de parsimonia y controlar el efecto engorde.

Los circuitos sintetizados son filtros analógicos, generalmente pasivos y en algún caso activos, considerando también transistores (Zebulum et al., 2000). El impacto del trabajo de Koza y sus colaboradores en este campo, como se describirá posteriormente, ha hecho que sus resultados se tomen como bancos de pruebas con los que realizar comparativas de los resultados obtenidos.

La evaluación de los circuitos, mediante simulación, es la tarea más costosa computacionalmente en electrónica evolutiva. Por lo que en el caso de circuitos pasivos, cuyos componentes son lineales (resistencias, condensadores y bobinas), generalmente se utilizan simuladores lineales realizados a medida. Sólo en el caso de circuitos activos, que son no lineales, se utiliza el simulador *SPICE* (Zebulum et al., 2000).

La evaluación de los filtros analógicos se realiza midiendo la respuesta en frecuencia del filtro en un número de de frecuencias y acumulando la desviación obtenida de la deseada. En algunos experimentos la evaluación se realiza en el dominio del tiempo, utilizando como especificación la respuesta del filtro al escalón unidad (Grimbleby, 2000).

Codificación de desarrollo

En relación con el método de codificación de desarrollo, la principal referencia son los trabajos de Koza y sus colaboradores (Koza et al., 1999a), donde consiguieron muy buenos resultados en la síntesis de circuitos mediante la electrónica evolutiva y que además han tenido un gran impacto en trabajos sucesivos, quedando sus problemas como banco de pruebas para nuevos algoritmos. Por otro lado, en relación con el presente trabajo, la aproximación de Koza merece especial atención y se describe con mayor detalle en la Sección 2.4.

Interacción implícita

Este tipo de representación se inspira en las redes de regulación genéticas (GRN) biológicas, por el que los genes disponen de zonas de regulación en la que reciben el mensaje de otros genes presentes en el cromosoma para regular su expresión. La representación Analog Genetic Encoding (AGE) introducida en (Mattiussi and Floreano, 2007), se basa en este modelo para la representación de circuitos electrónicos analógicos, pero también permite la representación de redes neuronales y redes de regulación genética, por lo que se ha propuesto su agrupación junto en lo que se llaman redes analógicas (Mattiussi and Floreano, 2007).

Un cromosoma agrupa un número variable de dispositivos cuyas conexiones se indican mediante un valor continuo que corresponde a la fortaleza de dicha conexión. Este tipo de valores corresponde a los pesos en el caso de redes neuronales, pero en circuitos analógicos corresponde a conductancias, por lo que finalmente se traducirán en resistencias.

Los circuitos sintetizados corresponden a problemas propuestos por Koza, si bien, se apartan del caso de filtros analógicos, obteniendo un circuito de referencia de voltaje, un circuito sensor de temperatura y un generador de función gaussiana. La evaluación de circuitos en el algoritmo evolutivo se realiza utilizando el simulador *SPICE*.

2.3.2. Trabajos previos en síntesis de circuitos electrónicos digitales

Aunque nuestro principal interés son los circuitos electrónicos analógicos, en este apartado se describen brevemente algunos trabajos previos en el ámbito de la síntesis de circuitos digitales, que finalmente se muestran agrupados en la Tabla 2.2.

En el ámbito de los circuitos digitales aparecen trabajos que utilizan una codificación directa de las puertas lógicas que componen un circuito combinacional, pudiendo representarse en forma matricial, en la que cada elemento correspondería a una puerta lógica. Esta representación puede ser genérica (Coello Coello and Aguirre, 2002), o bien corresponder a la codificación utilizada en una FPGA comercial (Yan and Jin, 2010), lo que en última instancia podría permitir su evaluación en el propio hardware.

Alternativamente a la representación de puertas, aparece la representación de productos lógicos de las variables de entrada de un circuito combinacional, cuyas sumas posteriores dan lugar a funciones booleanas. Por lo que el cromosoma permite codificar realmente funciones booleanas que posteriormente se pueden convertir a un circuito (Zebulum et al., 2000).

Finalmente, una tercera aproximación a la síntesis de circuitos digitales sería mediante la generación directa de código en un lenguaje de descripción de hardware como es Verilog. El código generado describiría un circuito que se puede implementar en un ASIC. En este caso el algoritmo evolutivo utilizado ha sido *Grammatical Evolution* (Karpuzcu, 2005).

Los circuitos digitales sintetizados son circuitos combinacionales como multiplexores o circuitos de paridad. Uno de los circuitos más habituales suele ser el sumador completo.

Referencia	Representación	Longitud	Año
(Zebulum et al., 2000)	Funciones booleanas	Variable	2000
(Coello Coello and Aguirre, 2002)	Puertas lógicas	Fija	2002
(Karpuzcu, 2005)	Código Verilog	Variable	2005
(Yan and Jin, 2010)	Puertas lógicas	Fija	2010

Tabla 2.2. – Trabajos previos en electrónica digital

2.4. Aproximación de Koza

Por la relevancia de esta aproximación para el desarrollo de este trabajo, en esta sección se describen las características de los trabajos realizado por Koza y sus colaboradores, haciendo uso de la Programación Genética, o PG , en el campo de la síntesis de circuitos analógicos (Koza et al., 1998; Koza et al., 1999b; Koza et al., 2000a; Koza et al., 2000b).

La aproximación de Koza tiene un especial interés por utilizar un nuevo tipo de algoritmo evolutivo como es la Programación Genética, junto con la obtención de muy importantes resultados en síntesis de circuitos electrónicos analógicos, en un ámbito mayor que otros trabajos anteriores generalmente centrados en la síntesis de filtros analógicos, y donde esta aproximación contempla, además del diseño de filtros, la síntesis de amplificadores, calculadores analógicos y el circuito de control de un robot sencillo. Koza afirma que los resultados obtenidos son comparables a los realizados por diseñadores humanos.

Como se indica en (Koza et al., 1999a), los circuitos electrónicos no son fácilmente representables en una estructura de árbol, ya que son grafos cerrados. Para este fin, Koza propone una codificación de desarrollo, en la que el cromosoma codifica una expresión de desarrollo que se compone de una serie de transformaciones. Estas transformaciones se aplican sobre un circuito básico de partida, llamado embrión, en el que existen conexiones modificables, que serán transformadas según indiquen las funciones, permitiendo alcanzar un embrión transformado. Este embrión se encuentra insertado en una estructura fija, que contiene la alimentación, la fuente de señal y las resistencias de entrada y de carga. A diferencia del embrión, la estructura fija no se modifica por las funciones de transformación. Su función es completar el circuito de forma que pueda funcionar como tal y realizar las pruebas y medidas oportunas sobre el mismo.

Las transformaciones o funciones se dividen en cinco tipos:

1. Funciones de creación de componentes
2. Funciones de modificación de la topología
3. Funciones de control del desarrollo
4. Funciones aritméticas para el cálculo de valores de los componentes

5. Funciones automáticamente definidas

Koza utiliza una gramática que define el lenguaje de las expresiones de desarrollo. Estas expresiones se pueden codificar de manera directa en un árbol de parseado, haciendo que el problema sea abordable mediante programación genética.

El proceso de evaluación de una solución conllevaría la aplicación de las transformaciones indicadas en la expresión de desarrollo correspondiente a dicha solución, sobre un embrión, para la obtención del circuito final. Dicho circuito se expresaría como lista de componentes o *netlist* que es la entrada del simulador de circuitos *SPICE* y finalmente se realizaría la simulación.

El resultado es un algoritmo que realiza la determinación de la topología y el dimensionamiento de los componentes a la vez.

Una de las complejidades que aparecen en PG es la creación de operadores de cruce y mutación que mantengan la consistencia de los árboles generados. Si bien esto es muy dependiente del lenguaje resultado en sí, concretamente de su gramática.

El operador de cruce más habitual propuesto en PG es la selección de un nodo aleatoriamente en cada árbol padre y el intercambio de los subárboles dependientes de cada nodo.

El operador de mutación más habitual es la selección de un nodo aleatoriamente del árbol y la sustitución del subárbol dependiente del mismo por un nuevo subárbol generado aleatoriamente.

En el caso de las expresiones de desarrollo, los operadores utilizados tienen en cuenta el tipo de subárbol, distinguiendo entre ramas del tipo *Result-Producing-Branch*, que aplican sobre una conexión o componente modificable, y ramas del tipo de generación de un valor aritmético.

Las funciones de adaptación utilizadas son muy dependientes del problema que se esté resolviendo, pues en cada caso los objetivos del circuito buscado son muy diferentes. Dichas funciones se suelen basar en diferentes medidas obtenidas sobre los circuitos tentativos, como por ejemplo, en el caso de síntesis de filtros analógicos se definen unos límites que son la frecuencia de corte y atenuación en las bandas eliminadas. Se evalúa el filtro con una señal sinusoidal en varias frecuencias y se mide la desviación del objetivo definido. Finalmente la adaptación se evalúa como la suma de las desviaciones en los puntos medidos, por lo que los mejores valores son los más bajos, haciendo que el problema sea de minimización. Otros problemas como pueden ser de cálculo analógico, discriminación de frecuencias o el controlador de un robot, requieren medidas y funciones de adaptación específicas de cada problema, siendo muy diferentes en cada caso.

Como se ha comentado previamente, la evaluación de todos los circuitos de la población se hace con el simulador *SPICE* y es costosa computacionalmente. La solución propuesta en (Koza et al., 1999a) se basa en paralelizar el algoritmo para su distribución en un cluster de 64 procesadores. También necesita una población muy elevada, definiendo poblaciones de 10,000 individuos en cada procesador.

3. *Grammatical Evolution*

Como se ha mencionado en el capítulo anterior son varias las aproximaciones utilizadas en electrónica evolutiva. La aproximación que se propone en este trabajo está basada en *Grammatical Evolution* (GE) debido a las interesantes ventajas que presenta tanto en codificación, utilizando cadenas binarias, como en el uso de operadores de variación estándar. Por ello este capítulo estará dedicado a describir este paradigma.

En la primera sección se describe el algoritmo *Grammatical Evolution* y a continuación se presenta un ejemplo de gramática simple y un ejemplo de decodificación de un cromosoma utilizando dicha gramática. Seguidamente se describen los operadores de recombinación, de mutación y el caso particular del operador de duplicación. Posteriormente se distinguen los casos particulares de genotipos inviables e inexpresables, y se comenta el efecto engorde, o *bloat,* y las formas más habituales de reducir su impacto.

3.1. Descripción general de GE

Grammatical Evolution, o GE, es un algoritmo evolutivo alternativo a la Programación Genética de Koza, que fue desarrollado por Michael O'Neill y Conor Ryan (Ryan and O'Neill, 1998; Ryan et al., 1998; O'Neill and Ryan, 1999a; O'Neill and Ryan, 1999b; O'Neill and Ryan, 2001; Ryan et al., 2002; O'Neill et al., 2003). GE es un algoritmo evolutivo capaz de generar código en cualquier lenguaje de programación, basado en cadenas binarias de longitud variable. A diferencia de PG que se basa en árboles de parseado para la representación del cromosoma. Los operadores de cruce y mutación actúan sobre las cadenas binarias. La decodificación del cromosoma se basa en una gramática BNF del lenguaje objetivo. Una gramática BNF se define mediante una tupla de 4 componentes, $\{S, T, N, R\}$ donde S es el símbolo de comienzo, T es el conjunto de terminales, N es el conjunto de no terminales o variables, y finalmente, R es el conjunto de las reglas de producción.

PG requiere el uso de operadores de cruce y mutación especialmente diseñados para el problema en cuestión, que deben operar sobre árboles de parseado y también deben garantizar la consistencia de los programas resultado. Esta consideración no es necesaria, siendo una de las ventajas más importantes, en el caso de GE, que sólo requiere el uso de operadores de cruce y mutación estándar. La consistencia de los programas generados se consigue mediante el proceso de decodificación.

GE utiliza cadenas de bytes, que se llaman codones, que es un nombre inspirado en la biología. Normalmente cada byte se considera un codón, disponiendo de 256 valores positivos para cada codón. La decodificación de una cadena determinada se realiza recorriéndola de izquierda a derecha, utilizando un codón por cada elección posible en las reglas de producción de la gramática del lenguaje. Para este proceso, las reglas de producción se numeran adecuadamente, de forma que cada paso de decodificación seleccionará una regla de producción. La Ecuación 3.1 determina la regla seleccionada a partir del valor del codón actual y del número de reglas posibles en el paso actual de la gramática. El uso del operador módulo garantiza que se obtendrá un valor adecuado para cualquier valor original del codón.

$$reglaSeleccionada = codón\, MOD\, númeroDeReglas \tag{3.1}$$

GE introduce el concepto de *wrapping*, basado en el fenómeno de superposición de genes observado en biología (O'Neill and Ryan, 2001). El mecanismo de *wrapping* consiste en comenzar de nuevo la cadena binaria una vez se ha recorrido de forma completa, habiendo agotado todos los codones de la misma. Si en el proceso de decodificación se llega al final de la cadena y son necesarios más codones para obtener una expresión final como decodificación completa del individuo.

3.2. Un ejemplo de gramática

Como problema inicial para nuestra implementación de GE se ha utilizado una gramática simple que permitirá posteriormente resolver problemas de regresión simbólica sencillos. La Tabla 3.1 muestra la gramática de expresiones simples en Extended Backus-Naur Form (ISO/IEC-14977, 1996). Esta gramática es muy similar a la mostrada en (O'Neill and Ryan, 2001).

A continuación se describe un ejemplo del procedimiento de decodificación. La cadena de ejemplo que se va a decodificar es: $[6, 8, 24, 27, 64, 8, 27, 3]$. Esta cadena se decodifica finalmente a la expresión $SIN(X + 1{,}0)$, y el proceso de decodificación es el siguiente:

1. La decodificación comienza con el símbolo de comienzo, S que corresponde a *expresión*
2. La regla de producción para la variable *expresión* tiene cuatro opciones y el primer codón vale 6. Aplicando la Ecuación 3.1, que da lugar a $6\, MOD\, 4$, obtenemos la regla 2, que a su vez se expande como: $función(expresión)$
3. La expansión de la regla de la variable $función$ con el siguiente codón, con valor 8, da lugar a la regla 0, que se expande como SIN, por lo que la expresión es ahora: $SIN(expresión)$

Tabla 3.1. – Gramática de expresiones simples

```
S =          expresión
T =          {'+', '-', '*', '/', 'SIN', 'COS', 'EXP', 'LOG', 'X', '1.0'}
N =          {expresión, operador, función, variable}

R = formado por las siguientes reglas de producción
expresión =  expresión, operador, expresión |            (0)
             '(', expresión, operador, expresión, ')' |  (1)
             función, '(', expresión, ')' |              (2)
             variable                                    (3)
operador =   '+' |                                       (0)
             '-' |                                       (1)
             '* ' |                                      (2)
             '/'                                         (3)
función =    'SIN' |                                     (0)
             'COS' |                                     (1)
             'EXP' |                                     (2)
             'LOG';                                      (3)
variable =   'X' |                                       (0)
             '1.0';                                      (1)
```

4. A continuación se determina el uso de la regla 0 para la variable *expresión* que da lugar a *expresión*, *operador*, *expresión*. Con ello la expresión es ahora: $SIN(expresión\, operador\, expresión)$
5. La regla 3 aplicada a la primera variable *expresión* se expande como *variable*, por lo que la expresión es ahora: $SIN(variable\, operador\, expresión)$
6. La regla 0 aplicada a la variable *variable* se expande como X, por lo que la expresión es ahora: $SIN(X\, operador\, expresión)$
7. La regla 0 aplicada a la variable *operador* da lugar a un signo más, por lo que la expresión es ahora: $SIN(X + expresión)$
8. La regla 4 aplicada a la variable *expresión* se expande como *variable*, por lo que la expresión es ahora: $SIN(X + variable)$
9. La regla 1 aplicada sobre la variable *variable* se expande como la constante 1,0, dando lugar a la expresión completa: $SIN(X + 1{,}0)$

3.3. Operadores de variación

En esta sección se describen los operadores de variación utilizados en GE, que como se ha indicado previamente, pueden ser operadores estándar utilizados en otros

algoritmos evolutivos. Se describen los operadores de recombinación, de mutación y el caso particular del operador de duplicación.

3.3.1. Operadores de recombinación

Estos operadores combinan la información contenida en dos cromosomas padre para obtener un nuevo cromosoma hijo. Este operador es propio de los algoritmos evolutivos y los distingue frente a otros paradigmas de búsqueda o de optimización.

Generalmente se parametriza con un valor de probabilidad de recombinación que suele variar entre 0,5 y 1,0 (Eiben and Smith, 2008), y que refleja la probabilidad de que los dos padres seleccionados se recombinen, o bien que se copien de forma directa.

Si bien existen diferentes tipos de operadores de recombinación en la literatura para algoritmos evolutivos, en GE se suele usar de forma general el operador de cruce de un punto tomado de los algoritmos genéticos, siendo el operador propuesto por defecto (O'Neill et al., 2003). En algoritmos genéticos en general este operador suele trabajar sobre cadenas de tamaño fijo, por lo que en el caso de GE es necesario adaptar este operador para su uso sobre cadenas de codones de tamaño variable.

El algoritmo de cruce de un punto para cadenas de tamaño variable funciona de la siguiente manera:

1. Se obtiene un número aleatorio para cada cadena padre, limitado por el tamaño de cada una.
2. Se intercambian la parte final de cada cadena, a partir del punto obtenido en el paso anterior.

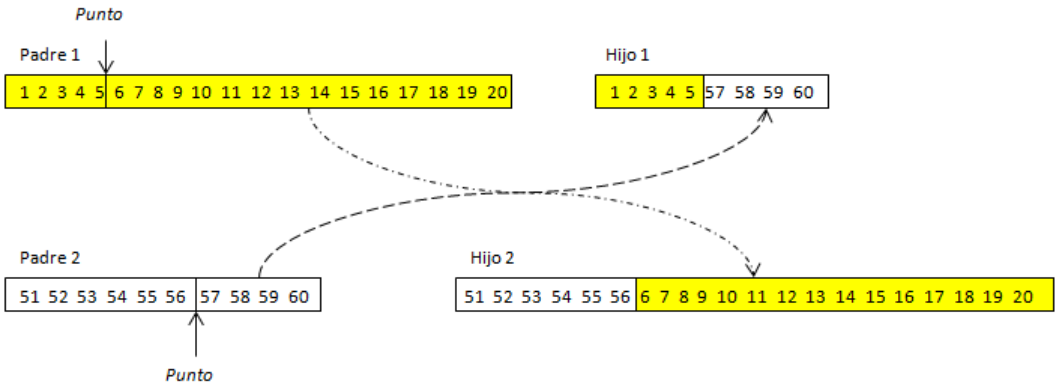


Figura 3.1. – Operador de cruce de 1 punto

3.3.2. Operadores de mutación

Los operadores de mutación trabajan sobre un cromosoma, generalmente obtenido como resultado del operador de recombinación, y producen una variación de forma aleatoria sobre el mismo. Se parametriza mediante una probabilidad cuya interpretación es propia del tipo de operador de mutación utilizado.

A diferencia del operador de recombinación, en GE no se propone un operador de mutación por defecto, por lo que a continuación se describen dos de los más habituales para cadenas de enteros, que son aplicables de forma directa a cadenas de bytes:

1. Operador *Random Reset*, es un operador de mutación que recorre una cadena de codones (bytes) de izquierda a derecha, y para cada codón obtiene un número aleatorio. En caso de que dicho número aleatorio sea menor que la probabilidad de mutación anteriormente indicada, el algoritmo generará un nuevo byte aleatoriamente que sustituirá al codón existente.
2. Operador de mutación *bitwise*, es un operador que recorre la cadena de codones bit a bit de izquierda a derecha y para cada uno de ellos obtiene un número aleatorio. En caso de dicho número aleatorio sea menor que la probabilidad de mutación anteriormente indicada, el algoritmo realizará el cambio (*flip*) del valor del bit considerado.

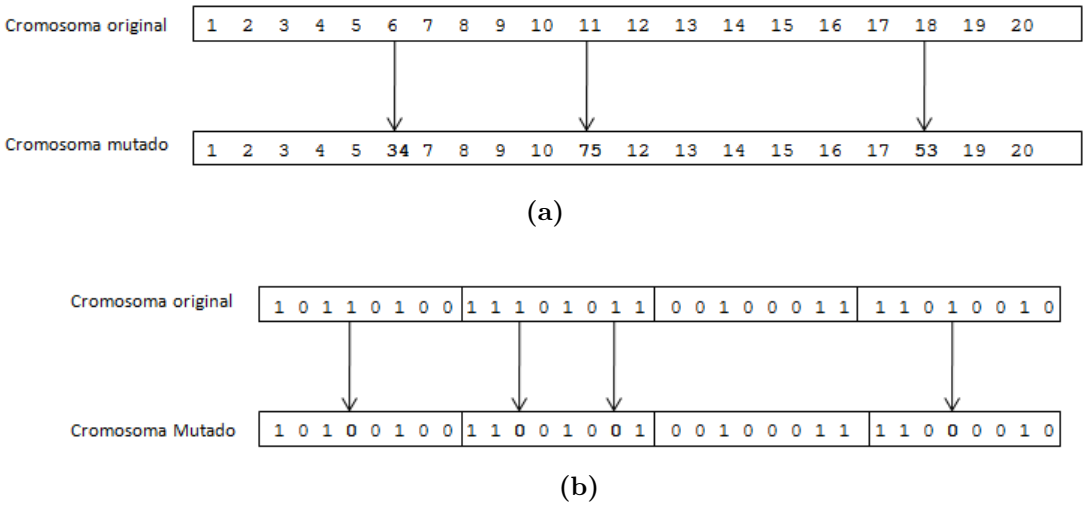


Figura 3.2. – Operadores de mutación (a) *random reset* (b) *bitwise*

3.3.3. Operador de duplicación

Este operador aumenta de tamaño una cadena resultado de los dos operadores anteriores, mediante la copia de parte de su contenido al final de la misma. La determinación de la parte de contenido que se va a copiar se realiza aleatoriamente, obteniendo

dos números aleatorios limitados por el tamaño actual de la cadena objeto de este operador. Este operador también viene parametrizado por una probabilidad de duplicación, que determina la probabilidad de que se aplique este operador o no.

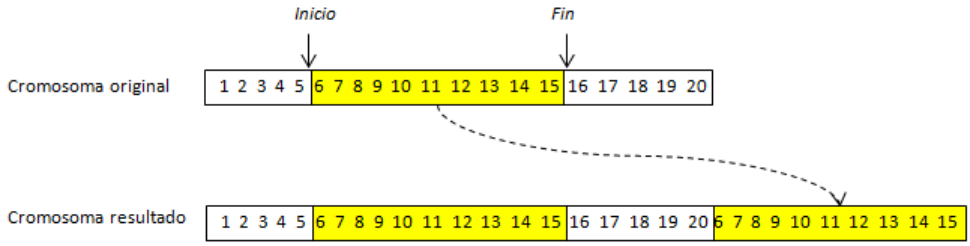


Figura 3.3. – Operador de duplicación

Este tipo de operador no es estándar para algoritmos evolutivos, sino que se introduce propiamente en GE (O'Neill and Ryan, 1999b; O'Neill and Ryan, 2001; O'Neill et al., 2003; O'Neill and Ryan, 2004), donde parece relacionado con la longitud variable de las cadenas binarias, orientado a introducir variabilidad en las longitudes de dichas cadenas. Sin embargo, se ha comprobado que no siempre se utiliza, pues en otras implementaciones de GE como GEVA (O'Neill et al., 2008) no se utiliza este operador, así como tampoco se utiliza en otros usos de GE (Karpuzcu, 2005).

3.3.4. Selección de padres

En cada generación es necesario seleccionar el conjunto de mejores individuos para que sean los padres de la siguiente generación. Esta selección se hace teniendo en cuenta los valores de adaptación de los individuos para que exista presión selectiva y el algoritmo tienda a producir mejores soluciones. Se comentan dos tipos de mecanismos de selección de padres muy habituales, si bien existen otros mecanismos posibles:

1. Selección por ruleta. Consiste en seleccionar cada padre aleatoriamente siendo la probabilidad de un individuo proporcional a su valor de adaptación. Este proceso se repite hasta conseguir el número de padres necesario. Este mecanismo presenta la ventaja de ser muy sencillo, pero también presenta varios problemas, pues en caso de presencia de individuos con una adaptación excepcional se puede producir una convergencia prematura. Por el contrario, en caso de que todos los individuos tengan adaptaciones similares puede ocurrir que apenas haya presión selectiva. Finalmente, en problemas de minimización es necesario transformar la adaptación de forma adecuada para uso de este mecanismo.
2. Selección por torneo. Consiste en la selección de un número de individuos de la población y la selección del que mejor valor de adaptación presente. Este proceso se repite hasta conseguir el número de padres necesario. El número

de individuos que componen el torneo es un parámetro de configuración del mecanismo, siendo muy habitual el torneo de 3 individuos. Este mecanismo presenta un mejor comportamiento que el de ruleta en los casos anteriormente indicado, por lo que es generalmente muy utilizado.

3.3.5. Selección de supervivientes

El modelo de selección de supervivientes, también llamado de reemplazo, define cómo se van reemplazando los individuos de la población en las generaciones sucesivas. Existen los dos siguientes modelos:

1. Modelo generacional. En cada generación se reemplazan todos los individuos por lo que es necesario crear una nueva población. Es un modelo de reemplazo basado en la edad de los individuos. Generalmente se suele utilizar conjuntamente con elitismo, que permite preservar los mejores individuos de una generación copiándolos directamente a la siguiente. El número de individuos copiados es un parámetro del elitismo.
2. Modelo *steady-state*. En cada generación sólo se reemplaza un número de individuos menor que el total de la población. Este número se llama gap, y en el modelo *steady-state* originalmente propuesto tomaba el valor 1. En este modelo se puede seguir una estrategia de reemplazo basada en edad, reemplazando los más viejos, o bien basada en adaptación, reemplazando los peores individuos de la población.

3.4. Genotipos inviables y genotipos inexpresables

En general en algoritmos evolutivos pueden aparecer genotipos cuyos genotipos correspondientes no se puedan evaluar correctamente. Por ejemplo en el caso de un problema de regresión simbólica de expresiones, podría aparecer una expresión dividida por cero, o bien, en el caso de síntesis de circuitos, podría aparecer un circuito sin conexión a la alimentación o con ésta cortocircuitada. Llamaremos a estos casos soluciones inviables o genotipos inviables. Normalmente estos genotipos se suelen penalizar con valores prefijados de adaptación que aseguren que no sean elegidos como padres, o bien, que la probabilidad de que sean escogidos sea muy baja.

En el caso de GE aparece una nueva clase propia de genotipos con problemas, que son aquellos cuyo proceso de decodificación no produce una expresión final. En este sentido GE introduce el mecanismo de *wrapping*, para permitir recorrer de nuevo una cadena de codones en caso de no haber terminado el proceso de decodificación, pero hay casos particulares que nunca producen una expresión. Por ejemplo, el proceso de decodificación de la cadena siguiente $[6, 8]$ sería el siguiente:

1. La decodificación comienza con el símbolo de comienzo, S que corresponde a *expresión*

2. La regla de producción para la variable *expresión* tiene cuatro opciones y el primer codón vale 6. Aplicando la ecuación Ecuación 3.1 $6\,MOD\,4$ obtenemos la regla 2 que se expande como: $función(expresión)$
3. La expansión de la regla de la variable $función$ con el siguiente codón, con valor 8, da lugar a la regla 0, que se expande como SIN, por lo que la expresión es ahora: $SIN(expresión)$
4. La cadena se ha terminado, pero utilizando el mecanismo de *wrapping* volvemos a empezar por la izquierda, obteniendo el codón de valor 6 que determina el uso de la regla 2 para la variable *expresión* que a su vez da lugar a $función(expresión)$. Con ello la expresión es ahora: $SIN(función(expresión))$
5. La expansión de la regla de la variable $función$ con el siguiente codón, con valor 8, da lugar a la regla 0, que se expande como SIN, por lo que la expresión es ahora: $SIN(SIN(expresión))$.
6. De nuevo se ha vuelto a terminar la cadena, por lo que volvemos a aplicar el mecanismo de *wrapping* volviendo a comenzar con el codón de valor 6, que se expande de forma similar a las veces anteriores dando lugar a la expresión: $SIN(SIN(función(expresión)))$
7. La siguiente expansión con el codón de valor 8, da lugar a la expresión: $SIN(SIN(SIN(expresión)))$
8. ....

Como se ve el proceso de decodificación continúa indefinidamente sin producir nunca una expresión final. Por este motivo es necesario añadir un límite al número máximo de veces que se aplica el mecanismo de *wrapping*, y una vez superado este límite, sin que se haya obtenido una expresión final, el proceso de decodificación se detendrá. El valor de límite máximo de *wrapping* es un parámetro más del algoritmo.

Definimos como genotipo inexpresable aquel cuya decodificación se detenga por haber superado el límite máximo de *wrapping.* Esta clase de genotipos recibe un valor de adaptación prefijado de penalización, de forma similar al caso de los genotipos inviables. En este caso, la penalización para los genotipos inexpresables es peor que en el caso de los inviables, pues se considera peor un genotipo que no da lugar a una expresión que el caso de un genotipo que da lugar a una expresión, pero cuyo fenotipo asociado es una solución inviable.

3.5. Efecto engorde

El efecto engorde o *bloat*, típico en Programación Genética (Eiben and Smith, 2008), y cuyo efecto es el crecimiento de los árboles de parseado a medida que aumenta el número de generaciones, se produce también en las aproximaciones basadas en GE.

Para reducir el efecto engorde, en PG, se pueden introducir medidas tales como imponer un tamaño máximo del árbol de parseado, pero las más utilizadas se basan en mecanismos de parsimonia que introducen presión selectiva para penalizar las soluciones basadas en árboles más grandes. Sin embargo y a diferencia de PG, en GE es posible introducir un mecanismo de limitación del tamaño máximo de las cadenas binarias, sin afectar a la consistencia de los programas generados, que se preserva mediante el proceso de decodificación.

4. Una nueva aproximación para el diseño de amplificadores electrónicos

Como se ha visto en el Capítulo 2, existen en la literatura diferentes propuestas que utilizan algoritmos evolutivos para la síntesis automática de circuitos electrónicos analógicos. La aproximación aquí presentada para abordar esta tarea, se basa también en el uso de algoritmos evolutivos. Concretamente, se fundamenta en la creación de una gramática que permite el diseño automático de circuitos analógicos y que está compuesta por un conjunto de funciones fuertemente inspiradas en las funciones de desarrollo de Koza (Koza et al., 1999a). Usando el enfoque evolutivo de la *Grammatical Evolution*, dicha gramática permitirá decodificar cada uno de los cromosomas de la población. El proceso de decodificación de un cromosoma implica obtener una serie de reglas de transformación, de acuerdo a la gramática, que se aplicarán a un circuito inicial, denominado embrión, para obtener, finalmente, el circuito final. El circuito embrión se compone de conexiones modificables sobre las que operan las transformaciones indicadas. Por otra parte, el embrión del circuito se considera embebido en una estructura fija, también llamada *test fixture*, que contiene el conjunto de componentes no evolucionables del amplificador, como son la alimentación, la fuente de señal, la resistencia de la fuente de señal y la resistencia de carga. Esta estructura fija, que no se modifica durante el proceso de desarrollo, permite completar el circuito de forma que pueda funcionar como tal y realizar las pruebas y medidas oportunas sobre el mismo.

A diferencia de la aproximación de Koza, que se basa en Programación Genética, la aproximación aquí propuesta está basada en *Grammatical Evolution*, por lo que en lugar de utilizar una representación basada en árboles de parseado, se utilizan cadenas lineales de bytes.

Aunque la gramática aquí propuesta está pensada para el diseño de amplificadores de una etapa, no debería resultar difícil su ampliación al diseño de amplificadores de más etapas o al diseño de cualquier otro tipo de circuito electrónico analógico como, por ejemplo, el diseño de filtros, uno de los tipos de circuitos más abordados en el diseño evolutivo de circuitos analógicos. Aunque la aproximación que se propone se podría usar para el diseño de cualquier tipo de circuito analógico, en este trabajo se ha utilizado para el diseño de amplificadores, ya que el diseño de filtros ha sido tratado con mayor atención en trabajos previos. Por otra parte, a diferencia de

los filtros que suelen ser circuitos pasivos, los amplificadores son circuitos activos y tienen requisitos particulares como pueden ser exigir una baja distorsión de la salida o una buena estabilidad con la temperatura.

Aunque existen gramáticas evolutivas que han sido utilizadas, desde el enfoque de la *Grammatical Evolution*, en el desarrollo automático de circuitos digitales (Karpuzcu, 2005), no se tiene constancia de que existan gramáticas evolutivas usadas en el diseño de circuitos analógicos. Por tanto, podría decirse que este trabajo es pionero en la creación de este tipo de gramáticas para el diseño automático evolutivo de este último tipo de circuitos.

En este capítulo se describe en primer lugar la gramática de las expresiones de desarrollo y las funciones de transformación utilizadas. Seguidamente se describe el proceso de desarrollo incluyendo un ejemplo de dicho proceso. Las siguientes secciones describen el algoritmo de simplificación y validación, así como la representación de un circuito mediante un grafo. Posteriormente se describen las especificaciones de un amplificador, y las medidas para comprobar dichas especificaciones, desde el punto de vista electrónico. A continuación se describe la función de adaptación utilizada en el algoritmo evolutivo. La siguiente sección describe los circuitos embrión utilizados y finalmente se incluyen unas consideraciones sobre la medida del margen dinámico.

4.1. Expresiones de desarrollo

En esta sección se describirá la gramática de las expresiones de desarrollo y las funciones de transformación que componen una expresión de desarrollo.

La Tabla 4.1 muestra la gramática utilizada para las expresiones de desarrollo en formato *Extended BNF* (ISO/IEC-14977, 1996). El símbolo de comienzo es *Expresion*, que produce una expresión de desarrollo completa. RPB significa *Result-Producing-Branch* según terminología de Koza y codifica una subexpresión que contiene las transformaciones que sufre una conexión o componente modificable determinado hasta el final del circuito.

El conjunto de funciones de transformación utilizado se ha compuesto partiendo de las funciones definidas por Koza (Koza et al., 1999a), con algunas variaciones. Por ejemplo, las funciones de conexión en serie o en paralelo se han simplificado evitando las duplicaciones de conexiones y componentes modificables existentes en el caso de la conexión en serie y se han simplificado de dos funciones a una en el caso de las conexiones en paralelo. También se ha introducido una función, no existente en el conjunto de Koza, para transformar un componente modificable en una conexión modificable.

Una expresión modificable comienza siempre por la función LIST seguido por un número de argumentos que determinan cada uno una *RPB* que aplica a una conexión

```
S =                 Expresion
T =                 {'LIST','END', 'CUT', 'NOP', 'PARALLEL', 'SERIES',
                    'THREE_GROUND', 'PAIR_CONNECT', 'THREE_VCC', 'R',
                    'C', 'WIRE', 'e', '0', '1',...,'9',
                    '-12','-11',...,'-3'}
N =                 {RPB, FUN0, FUN1, FUN2, FUN3, CC, ValorResistencia,
                    ValorCondensador,
                    digito,digitoNoCero,expCondensador}
R = formado por las siguientes reglas de producción
Expresion =         'LIST', '(', RPB, ')', { '(', RPB, ')' };
RPB =               FUN0 | FUN1, '(', RPB, ')' | FUN2, '(', RPB, ')',
                    '(', RPB, ')' | FUN3, '(', RPB, ')', '(', RPB,')',
                    '(', RPB, ')' | CC, '(', RPB, ')';
FUN0 =              'END' | 'CUT';
FUN1 =              'NOP' | 'WIRE';
FUN2 =              'PARALLEL';
FUN3 =              'SERIES' | 'THREE_GROUND' | 'PAIR_CONNECT' |
                    'THREE_VCC';
CC =                ( 'R', ValorResistencia, | 'C', ValorCondensador );
valorResistencia =  digitoNoCero, '.', digito, 'e', digito;
valorCondensador=   digitoNoCero, '.', digito, 'e', expCondensador;
digito =            '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'
                    | '9';
digitoNoCero =      '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' |
                    '9';
expCondensador =    '-12' | '-11' | '-10' | '-9' | '-8' | '-7' | '-6' |
                    '-5' | '-4' | '-3';
```

Tabla 4.1. – Gramática de expresiones de desarrollo

modificable del embrión. Por lo tanto, el número de argumentos de la función LIST vendrá indicado por el embrión utilizado, pues coincide con el número de conexiones modificables del mismo.

La Tabla 4.2 muestra la lista completa de funciones de transformación utilizadas en el algoritmo propuesto.

A continuación se explican con mayor detalle las funciones de transformación utilizadas, agrupadas mediante la taxonomía de Koza: funciones de creación de componentes, funciones de modificación de la topología y funciones de control del desarrollo.

Funciones de desarrollo	Nº argumentos	Descripción resumida
LIST	Variable	Comienzo de expresión, no se cuenta como función de transformación
R	2	Crea una resistencia sobre una conexión o componente modificable
C	2	Crea un condensador sobre una conexión o componente modificable
WIRE	1	Transforma un componente modificable en una conexión modificable
PARALLEL	2	Crea una nueva conexión modificable en paralelo
SERIES	3	Crea dos nuevas conexiones modificables en serie
THREE_GROUND	3	Copia la conexión o componente modificable en serie, y crea una conexión adicional a masa
THREE_VCC	3	Copia la conexión o componente modificable en serie, y crea una conexión adicional a alimentación
PAIR_CONNECT	3	Copia la conexión o componente modificable en serie, y crea una conexión adicional a otra parte del circuito
CUT	0	Elimina un componente o conexión modificable, dejándolo en circuito abierto
END	0	Indica el final de las transformaciones de un componente o conexión modificable
NOP	1	No realiza transformación. El componente o conexión modificable permanece siendo modificable

Tabla 4.2. – Funciones de desarrollo

4.1.1. Funciones de creación de componentes

Función R

Esta función crea una resistencia en la conexión modificable sobre la que actúa. En caso de que fuera un componente modificable, dicho componente se elimina y se crea en su lugar la resistencia indicada por esta función. El valor de la resistencia vendrá dado por el primer argumento de la función. La resistencia creada permanece siendo modificable, por lo que sucesivas transformación podrían cambiarla de forma que esta función quedara sin efecto.

Esta función lleva dos argumentos: el primero contiene el valor de la resistencia y el segundo es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a este componente modificable.

La Figura 4.1 muestra el resultado de esta función dependiendo de si opera sobre una conexión modificable, un condensador modificable o una resistencia modificable.

(a) **(b)** **(c)**

Figura 4.1. – Función R

Función C

Esta función crea un condensador en la conexión modificable sobre la que actúa. En caso de que fuera un componente modificable, dicho componente se elimina y se crea en su lugar el condensador indicado por esta función. El valor del condensador vendrá dado por el primer argumento de la función. El condensador creado permanece siendo modificable, por lo que sucesivas transformación podrían cambiarlo de forma que esta función quedara sin efecto.

Esta función lleva dos argumentos: el primero contiene el valor del condensador y el segundo es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a este componente modificable.

La Figura 4.2 muestra el resultado de esta función dependiendo de si opera sobre una conexión modificable, una resistencia modificable o un condensador modificable.

(a) **(b)** **(c)**

Figura 4.2. – Función C

Función WIRE

Esta función transforma el componente modificable sobre el que opera y lo convierte en una conexión modificable. En caso de que fuera inicialmente una conexión modificable no tendría efecto, actuando como la función NOP.

Esta función lleva un argumento que es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a esta conexión modificable.

La Figura 4.3 muestra el resultado de esta función dependiendo de si opera sobre una resistencia modificable, un condensador modificable o una conexión modificable.

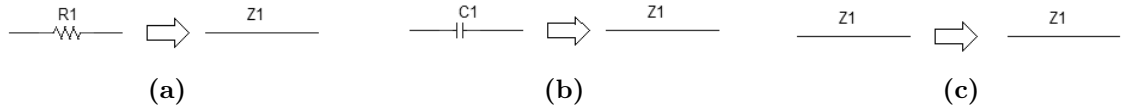


Figura 4.3. – Función WIRE

4.1.2. Funciones de modificación de la topología

Función SERIES

Esta función crea dos nuevas conexiones modificables, que se conectan en serie a la conexión o componente modificable inicial sobre el que opera la función. La conexión o componente inicial queda en el centro. De esta forma se evita distinguir entre una conexión serie por la derecha o por la izquierda, pues se agrupan ambas formas en una única función. Esta función es algo más simple que la propuesta por Koza, que duplicaba la conexión o componente modificable inicial y creaba una conexión modificable conectando los tres elementos en serie.

Esta función lleva tres argumentos correspondientes al componente y las dos nuevas conexiones modificables. Cada argumento es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a cada conexión o componente modificable.

La Figura 4.4 muestra el resultado de esta función dependiendo de si opera sobre una conexión modificable, una resistencia modificable o un condensador modificable.

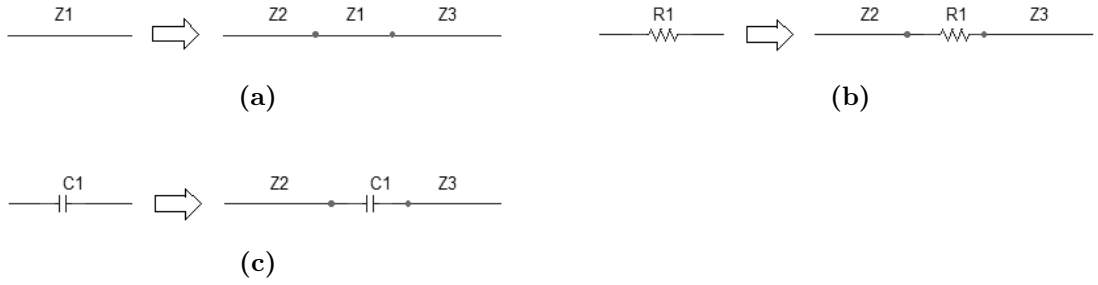


Figura 4.4. – Función SERIES

Función PARALLEL

Esta función conecta en paralelo una nueva conexión modificable a la conexión o componente modificable sobre la que opera. Esta implementación es más simple que la propuesta por Koza, que además constaba de dos variantes de conexión en paralelo.

Esta función lleva dos argumentos que corresponden a la conexión o componente modificable inicial y a la nueva conexión modificable creada. Cada argumento es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a cada conexión o componente modificable.

La Figura 4.5 muestra el resultado de esta función dependiendo de si opera sobre una conexión modificable, una resistencia modificable o un condensador modificable.

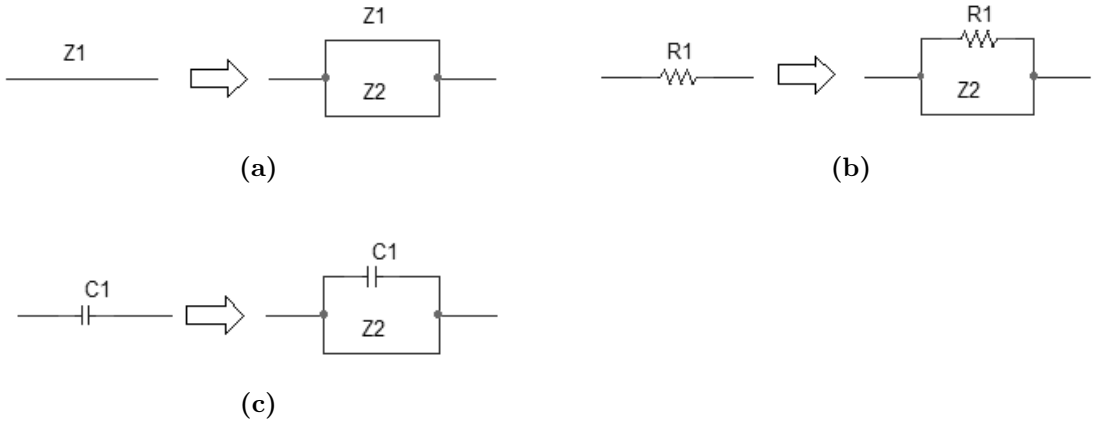


Figura 4.5. – Función PARALLEL

Función THREE_GROUND

La función THREE_GROUND duplica el componente o la conexión modificable sobre el que opera, realizando una copia de la misma y conectando ambas en serie. También crea una nueva conexión modificable que queda conectada al nodo central anterior. El otro nodo de la nueva conexión creada se conecta a masa.

Esta función lleva tres argumentos que corresponden a las tres conexiones o componentes modificables resultado de la misma. Cada argumento es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a cada conexión o componente modificable.

La Figura 4.6 muestra el resultado de esta función dependiendo de si opera sobre una resistencia modificable, un condensador modificable o una conexión modificable.

Función THREE_VCC

Esta función es muy similar a la función THREE_GROUND, pues también duplica el componente o la conexión modificable sobre la que opera, realizando una copia de la misma y conectando ambas en serie. También crea una nueva conexión modificable que queda conectada al nodo central anterior. A diferencia de THREE_GROUND,

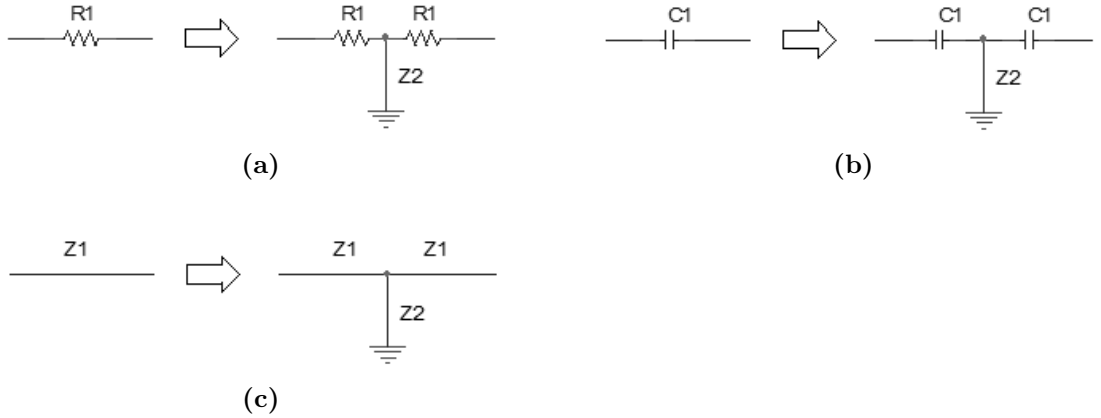


Figura 4.6. – Función THREE_GROUND

esta función conecta el segundo nodo de la nueva conexión modificable creada a la alimentación en lugar de a masa.

Esta función lleva tres argumentos que corresponden a las tres conexiones o componentes modificables resultado de la misma. Cada argumento es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a cada conexión o componente modificable.

La Figura 4.7 muestra el resultado de esta función dependiendo de si opera sobre una resistencia modificable, un condensador modificable o una conexión modificable.

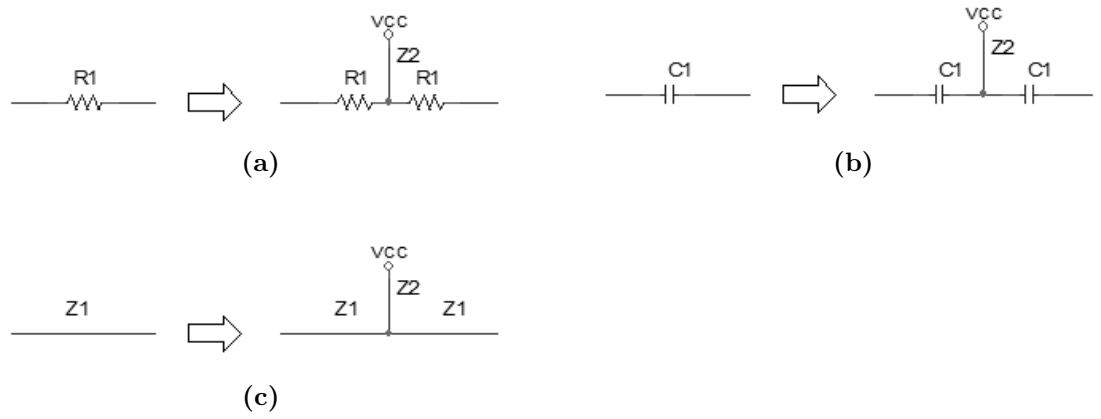


Figura 4.7. – Función THREE_VCC

Función PAIR_CONNECT

Esta función es muy similar a THREE_GROUND y THREE_VCC, pues también crea un duplicado del componente o la conexión modificable sobre la que opera,

quedando ambas conectadas en serie. Crea una nueva conexión modificable que se conecta al nodo central. El otro nodo de esta nueva conexión modificable se conectará a su homólogo creado en la siguiente función PAIR_CONNECT que se ejecute. Esta función permite crear conexiones entre dos puntos, que pueden ser distantes, del circuito.

En caso de que no se llame a una segunda función PAIR_CONNECT, la nueva conexión modificable quedaría sin conexión, por lo que no tendría efecto sobre el circuito y la conexión o componente que hubiera quedado al aire sería eliminado en la fase de simplificación del mismo.

Esta función lleva tres argumentos que corresponden a las tres conexiones o componentes modificables resultado de la misma. Cada argumento es una RPB que contiene todas las transformaciones sucesivas que se aplicarán posteriormente a cada conexión o componente modificable.

La Figura 4.8 muestra el resultado de esta función dependiendo de si opera sobre una resistencia modificable, un condensador modificable o una conexión modificable.

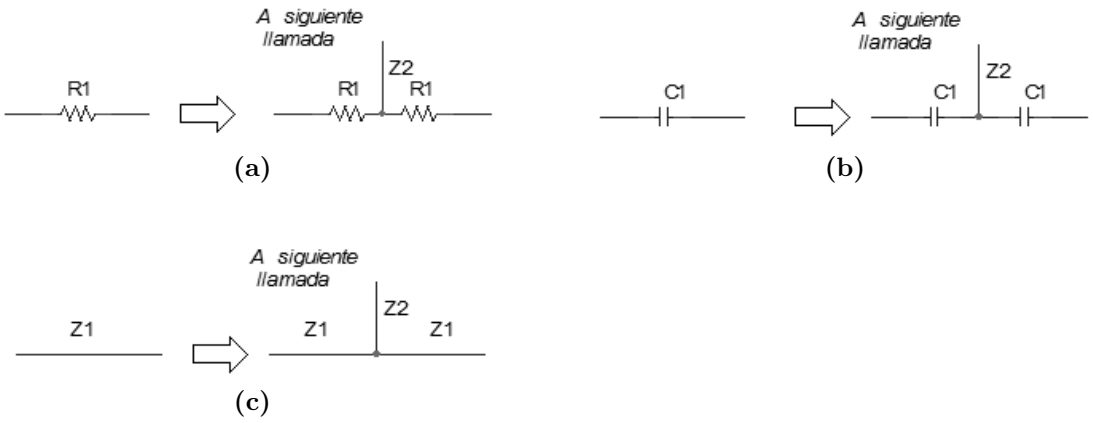


Figura 4.8. – Función PAIR_CONNECT

Función CUT

Esta función elimina el componente o conexión modificable sobre el que opera, quedando los dos nodos conectados en circuito abierto. Esta función no tiene argumentos, pues no hay nuevas transformaciones al no existir conexión o componente modificable.

4.1.3. Funciones de control del desarrollo

Función END

Esta función impide futuras transformaciones en el componente o conexión modificable sobre el que actúa, haciendo que deje de ser modificable. No tiene argumentos al no haber nuevas transformaciones que actúen sobre el componente o conexión modificable.

Función NOP

Esta función no realiza ninguna transformación sobre la conexión o componente modificable sobre el que opera, que además continúa siendo modificable. Su propósito es introducir un retardo en el desarrollo mientras se desarrollan otras partes del circuito. Puede tener un impacto en la topología cuando existen funciones de conexión entre partes del circuito como PAIR_CONNECT.

Esta función lleva un argumento que es una RPB. Como la conexión o componente modificable sobre la que opera continúa siendo modificable, las transformaciones sucesivas se incluyen en dicha RPB.

4.2. El proceso de desarrollo

En esta sección se describe cómo se procesa una expresión de desarrollo y cómo se van llamando a las diferentes funciones que actúan sobre el circuito embrión. También se muestra un ejemplo de decodificación de una cadena binaria hasta alcanzar el circuito final.

Como se ha comentado en secciones anteriores, el proceso de desarrollo opera sobre un circuito embrión formado por uno o más circuitos o componentes modificables. Una conexión modificable es una conexión de circuito que puede ser modificada durante el proceso de desarrollo para convertirse en un componente electrónico o bien en una conexión final. Un componente modificable, de la misma manera, también puede ser modificado durante el proceso de desarrollo para convertirse en otro componente electrónico diferente en tipo o valor, en un componente final o bien en una conexión modificable.

La serie de transformaciones sucesivas que sufrirá un circuito embrión vienen codificadas en el cromosoma, cuya decodificación produce una expresión de desarrollo asociada. Esta expresión se procesa mediante un parser cuya salida es un árbol de parseado. Este árbol tendrá una raíz en la que se encontrará la función LIST y de la que derivarán tantos argumentos como conexiones modificables tenga el embrión utilizado. Cada nodo del árbol tendrá una función de transformación y de él dependerán tantos nodos como argumentos tenga la función almacenada en dicho nodo.

Por ello, las hojas del árbol estarán formadas exclusivamente por funciones que no tengan argumentos. En la Tabla 4.3 se muestra un ejemplo de expresión de desarrollo y su árbol de parseado asociado. Este tipo de árbol se llama árbol de sintaxis abstracta (AST, del inglés *abstract syntax tree*) y representa la estructura sintáctica simplificada de un determinado lenguaje de programación, en nuestro caso de la expresión de desarrollo.

(a)

LIST (C 2.0e-9 (END)) (THREE_VCC(END)(CUT)(END)) (R 1.0e4(END)) (R 5.7e3(END))

(b)

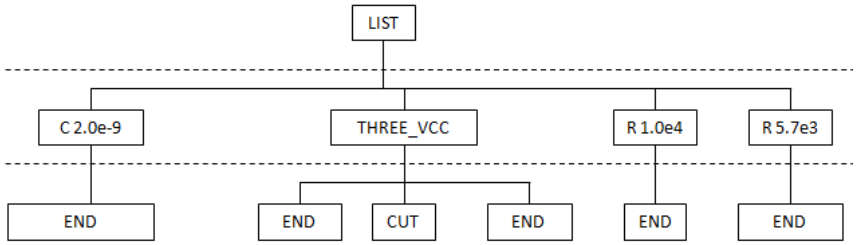


Tabla 4.3. – Ejemplo de expresión (a) y su árbol de parseado asociado (b)

El proceso de desarrollo consiste en la aplicación de las funciones almacenadas en las ramas del árbol de forma sucesiva. Cada aplicación de una función da lugar a un nuevo subcircuito, sobre el que opera la siguiente función hasta el final del proceso. El recorrido del árbol para determinar el orden de aplicación de las funciones es primero en anchura, o *breadth-first,* y en cada nodo se aplica la función que contiene a su conexión o componente modificable asignado. El recorrido en anchura conlleva que se aplicarán todas las funciones de un nivel del árbol antes de pasar al siguiente.

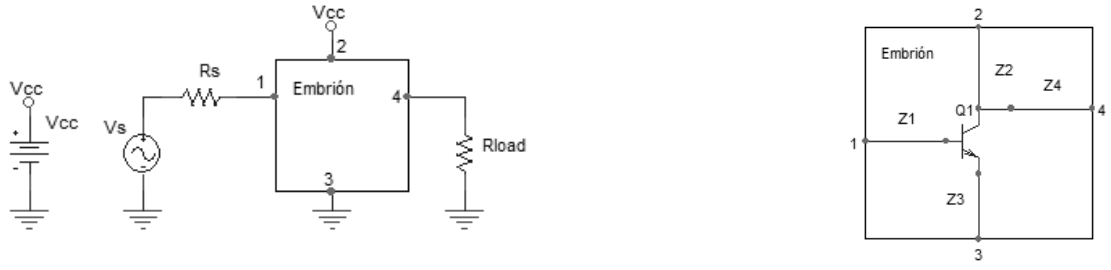
El circuito obtenido al recorrer un árbol puede depender del orden seguido para recorrerlo, debido a la existencia de funciones que permiten la conexión entre diferentes partes del circuito, como PAIR_CONNECT, que además dependen fuertemente del orden de ejecución de las mismas. Existen también funciones que no realizan ninguna acción en el momento en que se aplican, pero pueden retrasar la ejecución de la conexión o componente modificable sobre la que actúan y producir, sin embargo, un cambio en la topología debido a un cambio de orden de ejecución de una función PAIR_CONNECT siguiente.

4.2.1. Circuito embrión y estructura fija

El objetivo de este trabajo es la síntesis de amplificadores analógicos. En todo momento se asumirá que el amplificador a diseñar se podrá implementar mediante una única etapa de transistor bipolar en configuración emisor común. Por ello, el circuito

embrión de construye con un único transistor fijo, que no podrá ser modificado por el proceso de desarrollo y sólo las conexiones al transistor podrán sufrir modificaciones. Es posible construir diferentes tipos de embriones, y a continuación se describe un posible embrión de ejemplo que nos permitirá ilustrar el proceso de desarrollo. Los embriones utilizados en la evaluación se describen posteriormente en la Sección 4.7. El embrión utilizado como ejemplo, que se muestra en la Figura 4.9b, contiene el transistor fijo y cuatro conexiones modificables indicadas como Z_1, Z_2, Z_3 y Z_4.

Una vez definido el embrión utilizado, éste se sitúa en una estructura fija o *test fixture*, mostrado en la Figura 4.9a, que contiene una fuente de señal, su resistencia de fuente asociada y la resistencia de carga. Esta estructura fija no se modifica durante el proceso de desarrollo y es común para los distintos embriones utilizados en este trabajo.



(a) **(b)**

Figura 4.9. – Estructura fija (a) y ejemplo de embrión en emisor común (b)

4.2.2. Ejemplo de desarrollo

Como ejemplo del proceso de desarrollo, se va a mostrar el desarrollo de un embrión como el definido en el apartado anterior, que contiene cuatro conexiones modificables con un transistor en configuración de emisor común, (ver Figura 4.9). La expresión de desarrollo de ejemplo se muestra en Tabla 4.4, donde los argumentos se han separado en líneas para mayor facilidad de interpretación. Esta expresión no se ha obtenido mediante el proceso evolutivo y se muestra únicamente a título ilustrativo.

```
LIST
(R 4.0e3(PAIR_CONNECT(END)(END)(R 4.4e4 (END))))
(PAIR_CONNECT(R 9.8e2 (END))(NOP(R 6.6e1 (END)))(R 2.5e2 (R 8.5e4 (END)))
(R 6.5e3 (END))
(R 2.7e8 (C 9.4e-6 (END)))
```

Tabla 4.4. – Expresión de desarrollo

La expresión se analiza mediante el *parser* y se obtiene el AST o árbol de sintaxis abstracta de la misma, que se muestra en la Figura 4.10. Este árbol se va a recorrer según la estrategia primero en anchura mostrando los resultados de las transformaciones al recorrer cada nivel. Para facilitar el ejemplo, los subíndices de las conexiones modificables se van a conservar cuando se transformen en componentes, de forma que si Z_i se transformara en un condensador, nos referiremos a éste como C_i.

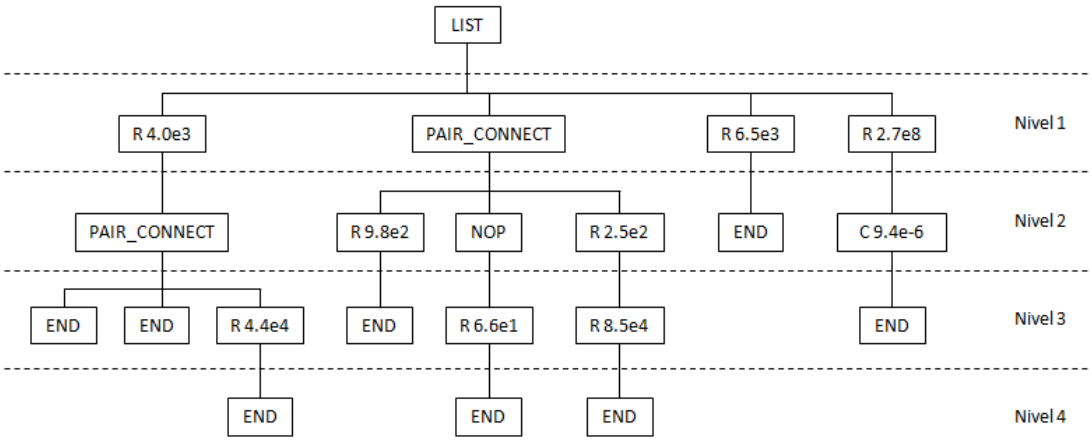


Figura 4.10. – AST

Nivel 0

En este paso todavía no se ha aplicado ninguna transformación, por lo que el circuito es directamente la estructura fija con el embrión conectado. Se muestra en la Figura 4.11.

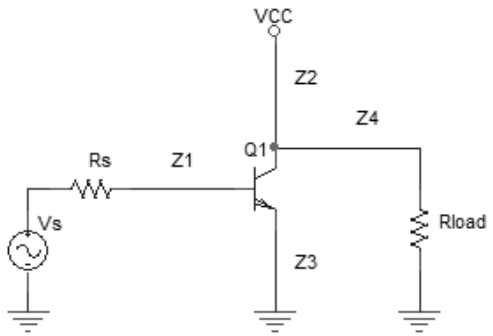


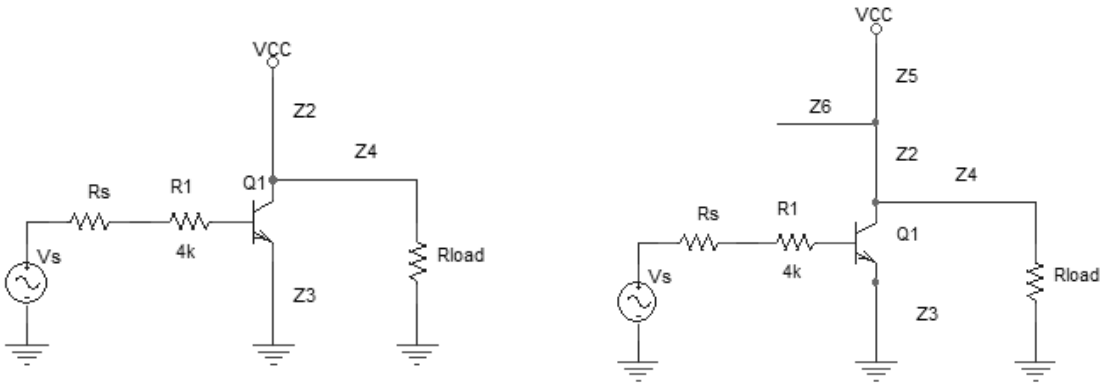
Figura 4.11. – Nivel 0: circuito embrión

Nivel 1

Se recorren todos los nodos del nivel 1 aplicando las transformaciones indicadas por las funciones de transformación.

1. Sobre la conexión modificable Z_1 se aplica una función R con un argumento 4,0e3, lo que da lugar a que se transforme en una resistencia de 4$k\Omega$. El circuito queda como se ve en la Figura 4.12a.
2. Sobre la conexión modificable Z_2 se aplica una función PAIR_CONNECT, por lo que se realiza una duplicación de la conexión Z_2, que da lugar a la nueva conexión modificable Z_5 y se conectan ambas en serie. Al nodo central de dicha conexión, se conecta una nueva conexión modificable Z_6. Esta última conexión se deja temporalmente con un nodo al aire, pendiente de conectar a la próxima función PAIR_CONNECT que se ejecute en la expresión. El circuito queda como se ve en la Figura 4.12b
3. Sobre la conexión modificable Z_3 se ejecuta una función R con un argumento 6.5e3 lo que da lugar a que se transforme en una resistencia de 6,5$k\Omega$. El circuito queda como se ve en la Figura 4.13a
4. La última transformación de este nivel se realiza sobre la conexión modificable Z_4, sobre la que se ejecuta una función R con un argumento 2,7e8, por lo que se ha transformado en una resistencia de 270$M\Omega$.

Al final del nivel 1, el circuito queda como se ve en la Figura 4.13b.



(a) **(b)**

Figura 4.12. – Nivel 1 (a) paso 1 (b) paso 2

Nivel 2

Se recorren todos los nodos del nivel 2 aplicando las transformaciones indicadas por las funciones de transformación.

1. Sobre la resistencia modificable R_1 se aplica una función PAIR_CONNECT, por lo que se realiza una duplicación de la misma que da lugar a R_7 y se

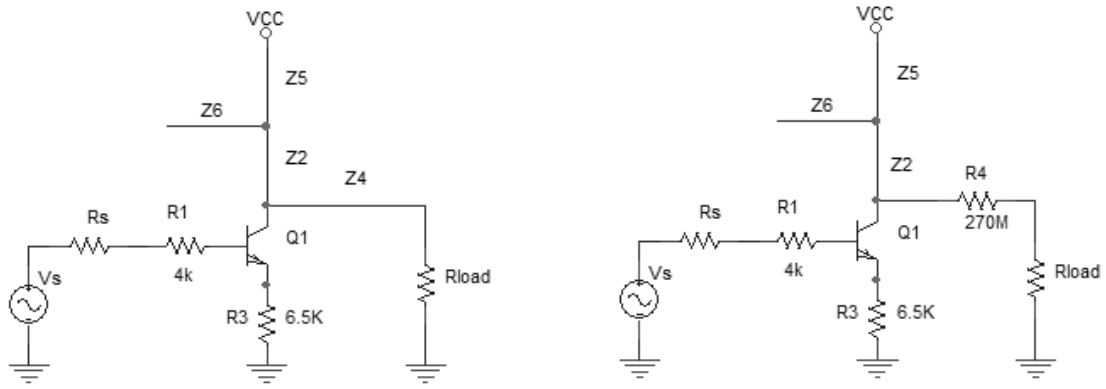


Figura 4.13. – Nivel 1 (a) paso 3 (b) paso 4

conectan ambas en serie. También se crea la nueva conexión modificable Z_8, y se conecta al nodo central de las resistencias R_1 y R_7. Al ser una segunda llamada a PAIR_CONNECT, el nodo libre de Z_8 se conecta al nodo al aire de Z_6. El circuito queda como se ve en la Figura 4.14a.

2. Sobre la conexión modificable Z_2 se ejecuta una función R con un argumento 9.8e2 se ha transformado en una resistencia de 980Ω. El circuito queda como se ve en la Figura 4.14b.
3. La conexión modificable Z_5 se ve afectada por una función NOP que no realiza ninguna transformación, quedando el circuito igual. El circuito permanece igual que en el paso anterior.
4. Sobre la conexión modificable Z_6 se aplica una función R con un argumento de 2.5e2 por lo que se transforma en una resistencia de 250Ω. El circuito queda como se ve en la Figura 4.15a.
5. Sobre la resistencia modificable R_3 se aplica la función END que termina las transformaciones de este componente modificable, por lo que este componente deja de ser modificable. El circuito permanece igual que en el paso anterior.
6. Sobre la resistencia modificable R_4 se aplica una función C con un argumento 9.4e-6 por lo que se transforma en un condensador de 9,4μF.

Al final del nivel 2, el circuito queda como se ve en la Figura 4.15b.

Nivel 3

Se recorren todos los nodos del nivel 3 aplicando las transformaciones indicadas por las funciones de transformación.

1. Sobre la resistencia modificable R_1 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito no se ve modificado a como se veía en la Figura 4.15b.

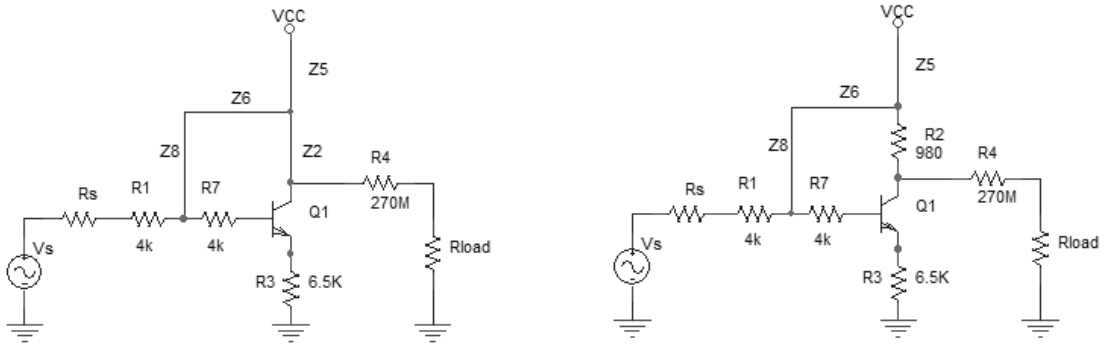


Figura 4.14. – Nivel 2 (a) paso 1 (b) paso 2

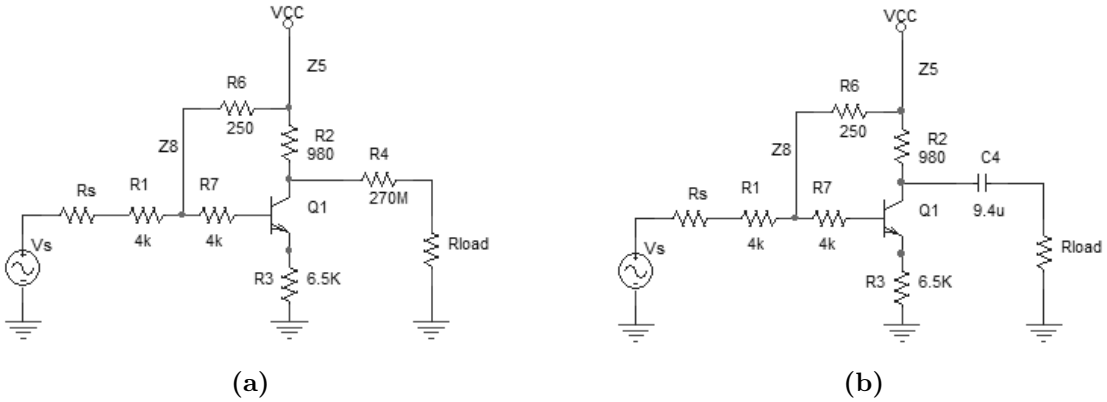


Figura 4.15. – Nivel 2 (a) paso 4 (b) paso 6

2. Sobre la resistencia modificable R_7 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito permanece igual que en el paso anterior.
3. Sobre la conexión modificable Z_8 se aplicación una transformación R con un argumento 4,4e4 por lo que se transforma en una resistencia de 44$k\Omega$.El circuito queda como se ve en la Figura 4.16a.
4. Sobre la resistencia modificable R_2 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito permanece igual que en el paso anterior.
5. Sobre la conexión modificable Z_5 se aplica una función R con un argumento 6,6e0 por lo que se transforma en una resistencia de 66Ω. El circuito queda como se ve en la Figura 4.16b.
6. Sobre la resistencia modificable R_6 se aplica una función R con un argumento 8,5e4 por lo que se transforma en una resistencia de 85$k\Omega$. El circuito queda como se ve en la Figura 4.16c.

7. Sobre el condensador modificable C_4 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito no se ve modificado a como se veía en la Figura 4.16c.

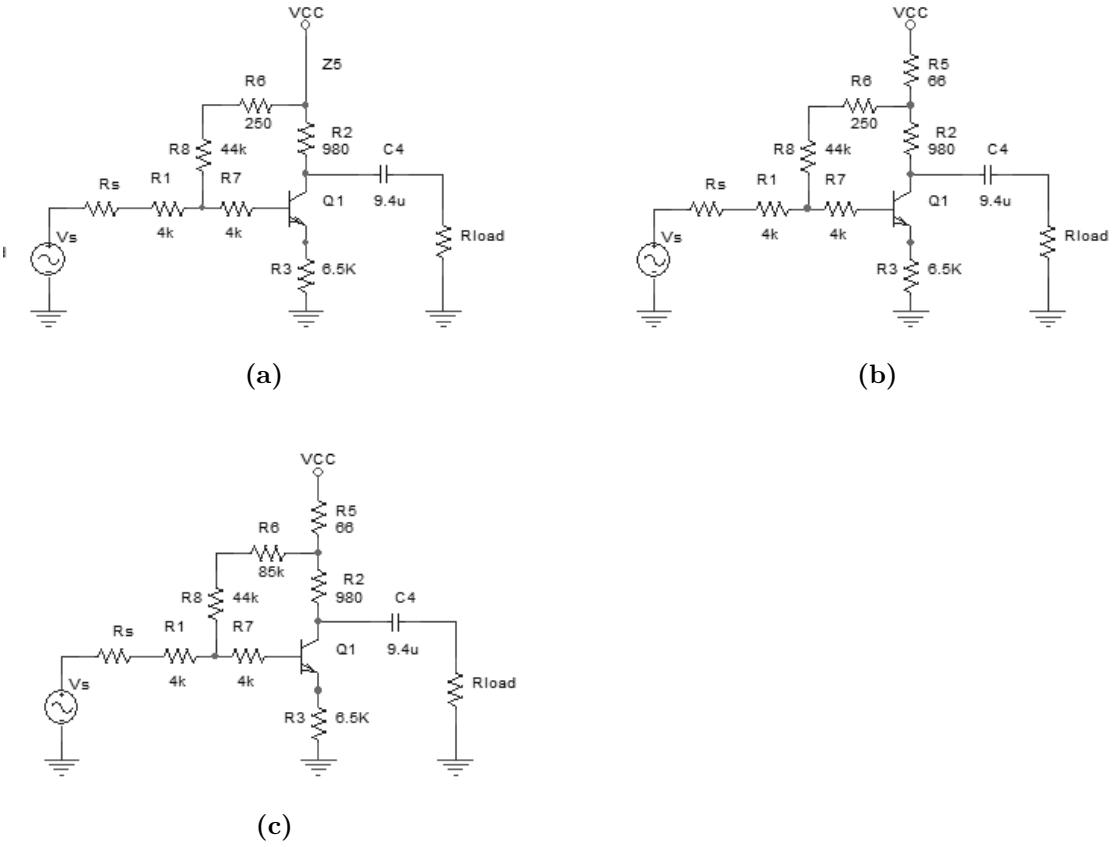


Figura 4.16. – Nivel 3 (a) paso 3 (b) paso 5 (c) paso 6

Nivel 4

Se recorren todos los nodos del nivel 4, el más profundo del árbol, aplicando las transformaciones indicadas por las funciones de transformación.

1. Sobre la resistencia modificable R_8 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito no se ve modificado a como se veía en la Figura 4.16c.
2. Sobre la resistencia modificable R_5 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito permanece igual que en el paso anterior.
3. Sobre la resistencia modificable R_6 se aplica la función END, que termina las transformaciones de este componente modificable, por lo que deja de ser modificable. El circuito permanece igual que en el paso anterior.

En este punto se han realizado todas las transformaciones indicadas en la expresión, por lo que el circuito final es el mostrado en la Figura 4.17, que se reproduce de nuevo, pues coincide con el mostrado en la Figura 4.16c.

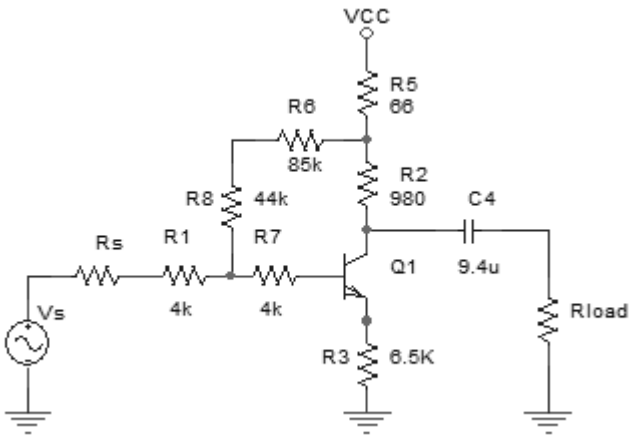


Figura 4.17. – Circuito final

4.3. Simplificación y validación de circuitos

En esta sección se describe un proceso de simplificación y validación de circuitos que se ejecuta una vez ha terminado el proceso de desarrollo, con el objetivo de eliminar componentes no funcionales y adecuar el circuito para su simulación. También se busca detectar circuitos que presenten problemas y no sean funcionales, con objeto de marcarlos como inviables.

Una vez ha terminado el proceso de desarrollo, se obtiene un circuito resultado que no puede ser simulado directamente, siendo necesario un proceso de simplificación. Por un lado es posible que hayan quedado conexiones modificables que no se hayan transformado en componentes. Estas conexiones, que son realmente un cable, no son directamente representables en el simulador de circuitos y deben ser simplificadas mediante el cortocircuito de sus terminales. En la Figura 4.18a se muestra un ejemplo en el que las conexiones modificables Z_1 y Z_2 no se han transformado finalmente en componentes, cuyos terminales se cortocircuitan dando lugar al circuito simplificado mostrado.

Otro caso que requiere simplificación es la existencia de componentes que sólo tengan un terminal conectado y que llamaremos componentes al aire. Estos componentes no aportan funcionalidad al circuito y pueden dar lugar a un error del simulador, por lo que se eliminan del circuito. En la Figura 4.18b se muestra un circuito resultado en el que ha quedado la resistencia R_3 al aire que se elimina en el circuito simplificado.

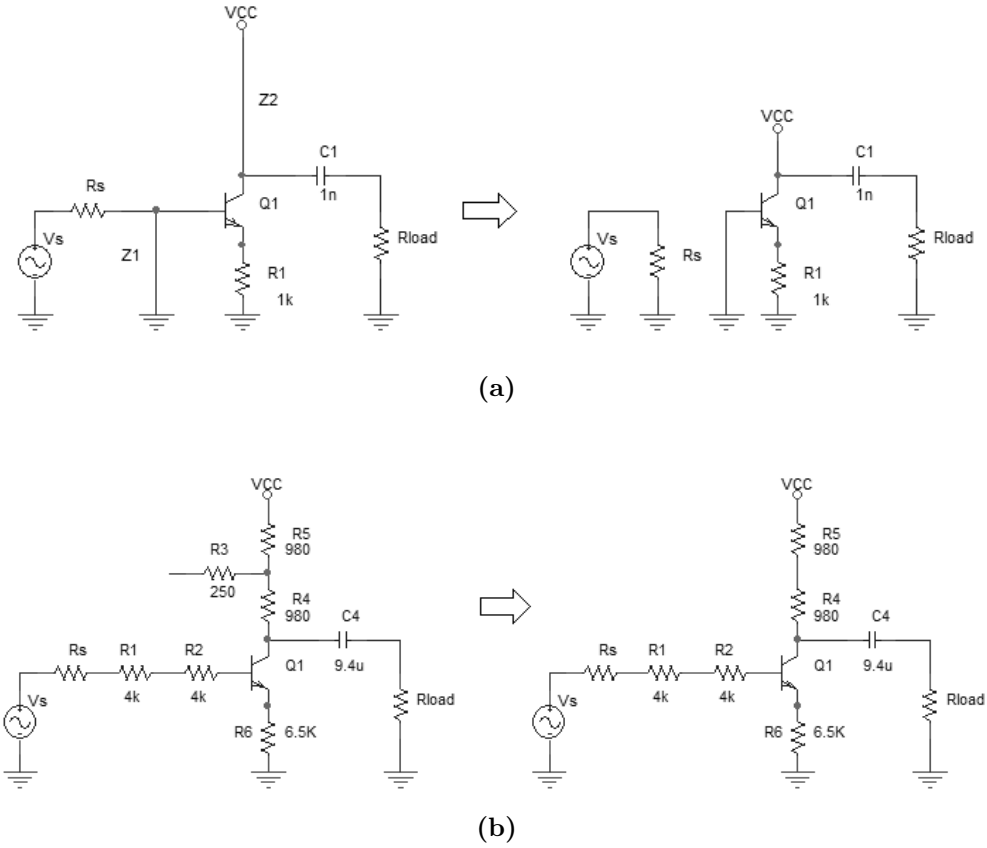


Figura 4.18. – Simplificación de circuitos (a) cortocircuito de nodos (b) componentes al aire

Como resultado del proceso de desarrollo y después de la simplificación, pueden darse varias circunstancias anómalas que hagan el circuito inviable. Por ejemplo, pueden aparecer circuitos en los que existan cortocircuitos en los generadores o en la resistencia de salida, así como se podrían producir circuitos en los que no haya conexión entre el generador de señal y la resistencia de salida. Este sería el caso mostrado en la Figura 4.18a, cuyo circuito simplificado no dispone de un camino entre la entrada y la salida, siendo un circuito inviable. Es interesante detectar los circuitos inviables mediante un proceso de validación, con objeto de ahorrar tiempo y evitar errores del simulador. Sin embargo, no siempre será posible detectar todas las situaciones anómalas, por lo que se seguirá una aproximación de eliminar aquellas que se puedan detectar de una forma sencilla, mientras que el resto de casos se detectarían mediante un análisis de la salida de error del simulador de circuitos.

4.4. Representación de circuitos y algoritmo de simplificación y validación

En esta sección se describe la estructura de datos que representa un circuito, sobre la que se realizan los procesos de desarrollo y de simplificación y validación. También se describe con mayor detalle el algoritmo de simplificación y validación de circuitos presentado en la Sección 4.3.

Un circuito se representará mediante un grafo cuyos arcos contienen componentes o conexiones. Estos componentes y conexiones pueden ser modificables o no. Los nodos se identificarán mediante un número que posteriormente se facilitará al simulador, por lo que es necesario que los número asignados respeten los requisitos de dicha herramienta, como por ejemplo, que el nodo de masa deba tener siempre la numeración cero.

La representación de un transistor, al ser un componente con tres terminales requiere un artificio, que consiste en considerar el transistor como un nodo del que salen tres arcos. Esta excepción requiere un tratamiento en la implementación. El nodo transistor recibirá un identificador de nodo, pero finalmente no se volcará al simulador, realizando la traducción adecuada para la entrada de dicho programa.

Cada circuito tentativo, comprendiendo estructura fija y embrión, se representa mediante un grafo y sobre el se realizará el proceso de desarrollo dando lugar a un grafo final. Este grafo será posteriormente sujeto al proceso de simplificación y validación.

Para evitar que en el proceso de desarrollo o en el de simplificación y validación se altere la estructura fija, es necesario protegerla frente a cambios, y para ello se define el concepto de nodo protegido. Si como resultado de los dichos procesos se elimina un nodo protegido, el circuito será marcado como inviable. Los nodos protegidos deberán ser aquellos pertenecientes a la estructura fija, por lo que comprenden el nodo de masa, el nodo (+) del generador de señal, el nodo (+) del generador de alimentación y los nodos de las resistencias del generador de señal y de la carga.

La forma de implementar un nodo protegido se ha realizado, de una forma sencilla, asignándoles los valores de nodo más bajos comenzando por masa 0, alimentación 1, señal 2, resistencia del generador de señal 3, resistencia de salida 4. La condición de inviabilidad de un circuito se cumplirá si una vez realizados los procesos de desarrollo y de simplificación y validación, se ha eliminado uno de los nodos con identificadores menores o iguales a 4.

En el algoritmo de simplificación y validación también se utiliza una propiedad de teoría de grafos, que indica que si un grafo tiene N nodos y menos de $N-1$ arcos no puede ser conexo, lo que daría lugar a un circuito inviable. Esta es una condición necesaria para que un grafo sea inconexo, pero que no garantiza que los grafos con más de N-1 arcos sean conexos. Sin embargo, puede detectar a priori algún circuito inviable y evitar la pérdida de tiempo de una simulación abocada al fracaso.

Finalmente, otro caso particular es la existencia de nodos de grado cero en el circuito. Esta condición parece inicialmente imposible de producir con el desarrollo de circuitos, sin embargo, la existencia de funciones como CUT que borran una conexión modificable, pueden dar lugar a estas situaciones.

Algoritmo de simplificación y validación

El algoritmo de simplificación y validación realizado es el siguientes

1. Simplificación de nodos cortocircuitados
    a) Búsqueda de todos las conexiones modificables que no se hayan transformado en componentes.
    b) Fusión de los nodos de cada conexión encontrada.
        1) Elegir el nodo de inferior valor de identificador, que permanecerá con objeto de preservar los nodos protegidos. El otro nodo se borrará al final de la operación de fusión de nodos. En caso de que ambos nodos fueran protegidos, el circuito se marca como inviable.
        2) Eliminar todos los arcos que hubiera entre ambos nodos, pues el cortocircuito de los nodos los hace no funcionales.
        3) Conectar al nodo elegido todos los arcos que terminaban en el nodo a borrar.
        4) Borrar el otro nodo
2. Eliminación de componentes al aire y de nodos de grado cero
    a) Se recorren todos los nodos del circuito y se comprueba su grado.
        1) Si el nodo es de grado cero,
            a′ En caso de que el nodo sea protegido, el circuito se marca como inviable
            b′ En caso de que no sea un nodo protegido, se borra.
        2) Si el nodo es de grado uno, representa un componente al aire.
            a′ Se busca el arco asociado a dicho nodo
            b′ Se eliminan el arco y el nodo. Existiendo un caso particular en el que no se elimina el arco, que corresponde a un terminal del transistor que hubiera quedado al aire.
3. Detección de si el circuito tiene asociado un grafo inconexo, comprobando si tiene N nodos y menos de N-1 arcos. En caso afirmativo, el circuito se etiqueta como inviable.

4.5. Especificaciones de diseño del amplificador

En esta sección se define, desde el punto de vista de la electrónica, el concepto de las distintas especificaciones implicadas en el diseño de un amplificador. Seguidamente, en la Sección siguiente, se mostrará el cómo hacer computable cada especificación y poder así construir una función matemática objetivo que permita medir de forma conjunta dichas especificaciones.

El objetivo de los experimentos será conseguir un amplificador de tensión de una etapa basado en un transistor bipolar. El transistor utilizado ha sido el $2N2222$ que es un transistor NPN, de baja potencia, de uso habitual en diseños electrónicos. El circuito constará también de resistencias y condensadores, sobre los que no se realizará la restricción de que tengan que corresponder con valores comerciales, por lo que se considerará que pueden tomar cualquier valor posible dentro de la gramática utilizada. Esta circunstancia se aplicará tanto en los diseños realizados por el algoritmo evolutivo como en los diseños realizados por el humano.

Las especificaciones del amplificador buscado contemplarán las siguientes características: ganancia, impedancia de entrada, impedancia de salida, frecuencia de corte inferior y caída de ganancia una década inferior, distorsión, polarización y estabilidad ante la temperatura.

La estructura fija constará de una fuente de alimentación, un generador de señal con una resistencia interna y una resistencia de carga.

Ganancia

Las medidas de ganancia se harán muestreando dicho valor a diferentes frecuencias, $f_1, \ldots, f_n$. situadas dentro del ancho de banda del amplificador a diseñar, siendo n dependiente de la magnitud de dicho ancho de banda (a mayor ancho de banda, mayor muestreo). A estas frecuencias se exige que la ganancia sea la establecida en las condiciones de diseño. Se obtienen las medidas de V_i y V_o, según la Figura 4.19a, para las frecuencias indicadas y la ganancia viene dada por la Ecuación 4.1.

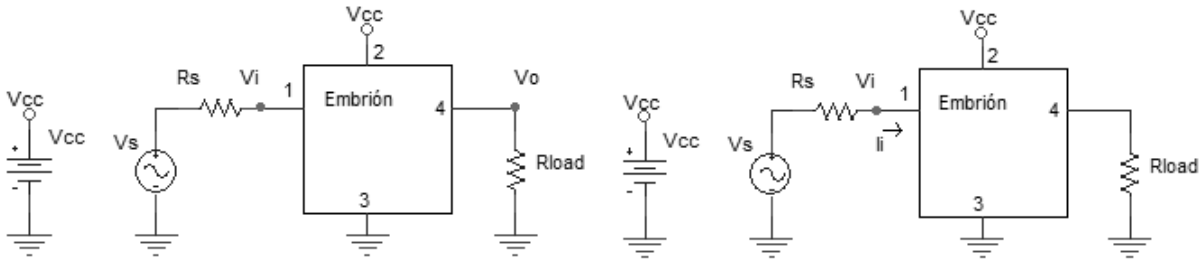
$$A_v(f_i) \mid_{dB} = 20\, log\left(\frac{V_o(f_i)}{V_i(f_i)}\right) \tag{4.1}$$

Impedancia de entrada

Las medidas de la impedancia de entrada se harán a la frecuencia f_{media} en la que se exigirá que la impedancia sea la establecida en las condiciones de diseño

especificadas. Se obtienen las medidas de V_i e I_i, según la Figura 4.19b, para las frecuencias indicadas, y la impedancia de entrada viene dada por la Ecuación 4.2.

$$Z_{in}(f_{media}) = \frac{V_i(f_{media})}{I_i(f_{media})} \tag{4.2}$$



(a) Medida de la ganancia **(b)** Medida de la impedancia de entrada

Figura 4.19.

Impedancia de salida

Para esta medida no resulta adecuada la estructura fija propuesta hasta el momento, por lo que es necesario cortocircuitar la fuente de señal y retirar la resistencia de carga, en cuyo lugar se pondrá una fuente de señal. Con ello, la estructura fija queda modificada como se indica en la Figura 4.20. Las medidas de la impedancia de salida se harán a la frecuencia f_{media} en la que se exigirá que la impedancia sea la establecida en las condiciones de diseño especificadas. Se obtienen las medidas de V_o e I_o, para la frecuencia f_{media} y la impedancia de salida viene dada por la Ecuación 4.3.

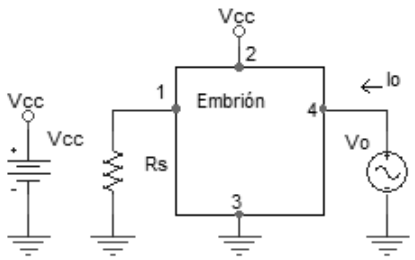


Figura 4.20. – Circuito para medida de la impedancia de salida

$$Z_{out}(f_{media}) = \frac{V_o(f_{media})}{I_o(f_{media})} \tag{4.3}$$

Frecuencia de corte inferior y caída de ganancia

La definición de frecuencia de corte es la mínima frecuencia a la que se produce una caída de ganancia de $3dB$ frente a la ganancia en la banda de paso. En las especificaciones se indica la frecuencia f_{low} que deberá ser la frecuencia de corte inferior. Adicionalmente se exige que para valores inferiores a la frecuencia de corte, la ganancia continúe cayendo a razón de una cantidad especificada por década. Para ambas medidas se obtendrá la ganancia del circuito en las frecuencias f_{low} y $f_{low}/10$, usando la expresión dada por la Ecuación 4.1, según la Figura 4.19a.

Corriente de colector y estabilidad de la misma frente a la temperatura

Para garantizar que el transistor se encuentra correctamente polarizado, se exige que tenga un valor dentro de un rango definido en las especificaciones, obtenido a partir de las especificaciones del transistor. En caso de que no se encuentre en dicho margen, se introducirá una penalización, mayor cuanto más alejada se encuentre la corriente de colector de dicho margen. El objetivo es evitar que esté muy cerca del corte, lo que daría lugar posteriormente a distorsiones, así como evitar un exceso de disipación de potencia, lo que podría dar lugar a que se quemara el transistor.

Hasta ahora todas las medidas se han obtenido a temperatura ambiente, sin embargo, es importante que el amplificador ofrezca una ganancia estable frente a cambios en la temperatura, lo que viene determinado por la estabilidad de la corriente de colector con la temperatura. Para ello se ha considerado realizar una medida del amplificador a una temperatura mayor especificada.

Medida de distorsión

Una de las especificaciones que utiliza un diseñador humano en la tarea de diseño de un amplificador es el margen dinámico, o también rango dinámico, buscando siempre obtener el máximo. Esta medida corresponde a la máxima señal sinusoidal que puede entregar el amplificador sin saturar, y es muy fácil de obtener en un laboratorio utilizando un osciloscopio conectado a la salida del amplificador y un generador de señal a la entrada del mismo. Así, el margen dinámico vendrá dado por la excursión pico-pico de la señal de salida.

Esta medida que es muy sencilla para un diseñador humano, resulta muy compleja de obtener de forma automatizada a partir de una señal, independientemente de si es muestreada de un circuito real o bien procedente de una simulación. Esta problemática, que se describe con mayor detalle en la Sección 4.8, ha dado lugar a que se abandonara la medida del margen dinámico en favor de una medida de distorsión.

Esta medida se basa en un análisis de Fourier sobre la señal de salida, que permite obtener las amplitudes de los armónicos de la frecuencia fundamental. Cuando se

introduce una señal sinusoidal pura a una frecuencia en un amplificador ideal, la salida ideal correspondería a una señal sinusoidal pura multiplicada por la ganancia y con un desfase frente a la señal de entrada. En el caso de que el amplificador no sea ideal, las no linealidades del mismo darán lugar a que aparezcan componentes armónicas, cuya frecuencia es múltiplo de la frecuencia fundamental de la señal de entrada. En el análisis de Fourier se observará que las amplitudes de dichos armónicos no valdrán cero.

Koza utiliza una aproximación similar en la síntesis de amplificadores, utilizando en un caso un análisis de Fourier de la salida del amplificador, obteniendo el valor de amplificación a partir de la amplitud de la componente fundamental y exigiendo que las amplitudes de los armónicos valgan cero. En otro caso utiliza directamente el valor de la distorsión armónica (THD, del inglés *Total Harmonic Distorsión*).

En este trabajo se utilizará la segunda aproximación, basada en la medida de la distorsión armónica, que relaciona la potencia de los armónicos frente a la potencia de la componente fundamental, de acuerdo con la Ecuación 4.4.

$$THD = \frac{\sqrt{\sum_{i=2}^{n} V_i^2}}{V_1} \tag{4.4}$$

Donde V_1 es la amplitud de la componente fundamental y los V_i para $i > 1$ son las amplitudes de los armónicos.

Esta medida se realizará utilizando una señal sinusoidal pura de frecuencia f_{media} y se exigirá que el valor de *THD* sea menor de un umbral especificado.

4.6. Función de adaptación

En la sección anterior, se definió el concepto, desde un punto de vista electrónico, de cada una de las distintas especificaciones implicadas en el diseño de un amplificador. Sin embargo, en el diseño automático de circuitos, hay que tener en cuenta dos limitaciones. La primera viene condicionada por el hecho de que la medida de cada una de las especificaciones no la realiza un humano y, la segunda, viene determinada por el hecho de que no sólo se debe realizar la medida sino, también, simultáneamente debe medirse cómo de cerca está la medida de la especificación de diseño dada. Salvar la primera limitación no es trivial ya que, en algunos casos, implica hacer computable un conjunto de pasos que, sin embargo, para el humano no reviste ningún tipo de dificultad. Para afrontar la segunda limitación, será necesario formalizar cuál es el error cometido en la medida de cada especificación. Esta sección se dedica, por tanto, a describir cuál ha sido la estrategia seguida para afrontar estos dos retos, todo ello, con el objetivo final de construir una función de adaptación que permita

al algoritmo evolutivo evaluar la bondad de un circuito (individuo) en relación a las especificaciones concretas de un determinado diseño.

En primer lugar se describirá la función de adaptación utilizada y seguidamente se describirán los diferentes términos que la componen, que se obtienen a partir de las medidas realizadas sobre el circuito simulado. Los términos utilizados corresponden a las diferencias de dichas medidas frente a los valores especificados para las mismas. Adicionalmente se han normalizado estos términos frente al valor especificado, con el objetivo de evitar el mayor peso de algunos términos frente a los demás. Finalmente, sobre estos términos se hace actuar una función de transformación, indicada por $T(\Delta x)$, para bien expandir o bien comprimir, el rango numérico de los mismos y así aumentar o disminuir la presión selectiva. La función de adaptación viene dada por la Ecuación 4.5.

$$F = \sum_{i=1}^{n} F_{A_v}[i] + F_{Z_{in}}[i] + F_{Z_{out}}[i] + \sum_{i=1}^{2} F_{f_l}[i] + \sum_{i=1}^{2} F_{I_C}[i] + F_{THD} \tag{4.5}$$

Finalmente el objetivo del problema será minimizar el valor de adaptación ofrecido por la expresión anterior.

A continuación se describe cada sumando de forma individualizada. De forma general se seguirá el criterio de indicar las especificaciones mediante variables sin tilde e indicar con tilde las medidas realizadas sobre el circuito simulado. Por ejemplo, A_v indicará la ganancia especificada mientras que $\widetilde{A}_v$indicará la ganancia medida en el circuito simulado.

Ganancia

El número de sumandos, n, vendrá determinado por el muestreo de frecuencias a las que se realizan las medidas, que dependen de las condiciones de especificación. Los sumandos correspondientes a la ganancia vienen indicados en la Ecuación 4.7.

$$\Delta_{A_v}[i] = \frac{\mid A_v - \widetilde{A}_v(f_i) \mid}{A_v}, i = 1, \ldots, n \tag{4.6}$$

$$F_{A_v}[i] = T(\Delta_{A_v}[i]) \tag{4.7}$$

Impedancia de entrada

La impedancia de entrada se mide a la frecuencia f_{media}, que es una frecuencia situada aproximadamente en la mitad del ancho de banda. El sumando correspondiente

a la impedancia de entrada viene indicado en la Ecuación 4.9.

$$\Delta_{Z_{in}} = \frac{\mid Z_{in} - \widetilde{Z}_{in}(f_{media}) \mid}{Z_{in}} \tag{4.8}$$

$$F_{Z_{in}} = T\left(\Delta_{Z_{in}}\right) \tag{4.9}$$

Impedancia de salida

La impedancia de salida se mide a la frecuencia f_{media}, que es una frecuencia situada aproximadamente en la mitad del ancho de banda. El sumando correspondiente a la impedancia de salida viene indicado en la Ecuación 4.11.

$$\Delta_{Z_{out}} = \frac{\mid Z_{out} - \widetilde{Z}_{out}(f_{media}) \mid}{Z_{out}} \tag{4.10}$$

$$F_{Z_{out}} = T\left(\Delta_{Z_{out}}\right) \tag{4.11}$$

Frecuencias de corte

La aproximación seguida en la medida de la frecuencia de corte no es la búsqueda de la misma, pues conllevaría la realización de un análisis en varias frecuencias buscando el valor de caída de ganancia de $3dB$. Por el contrario se ha optado por realizar una medida a la frecuencia de corte especificada f_{low}, y penalizar la desviación de este valor del especificado. Esta aproximación resulta menos costosa computacionalmente, pues sólo requiere la simulación del circuito a una frecuencia, frente a varias. Como se ha indicado anteriormente, las medidas de ganancia se realizan a las frecuencias f_{low} y $f_{low}/10$. Los sumandos correspondientes a la impedancia de salida vienen indicados en la Ecuación 4.13 y la Ecuación 4.15.

$$\Delta_{f_l}[1] = \frac{\mid (A_v - 3dB) - \widetilde{A}_v(f_{low}) \mid}{A_v - 3dB} \tag{4.12}$$

$$F_{f_l}[1] = T(\Delta_{f_l}[1]) \tag{4.13}$$

$$\Delta_{fl}[2] = \frac{\mid (A_v - 3dB - caída) - \widetilde{A}_v\,(f_{low}/10) \mid}{A_v - 3dB - caída} \tag{4.14}$$

$$F_{fl}[1] = \begin{cases} 0 & Si\ \widetilde{A}_v\,(f_{low}/10) \leq (A_v - 3dB - k) \\ T(\Delta_{fl}[1]) & en\ otro\ caso \end{cases} \tag{4.15}$$

Donde k es el valor de caída en dB por década exigido en las especificaciones.

Corriente de colector y estabilidad frente a la temperatura

La corriente de colector se mide a temperatura ambiente, T_a, y a una temperatura superior, T_c. El sumando correspondiente al ajuste de la corriente de colector al margen especificado, viene indicado en la Ecuación 4.17 y el sumando correspondiente a la estabilidad de la corriente de colector con la temperatura viene indicado en la Ecuación 4.19.

$$\Delta_{I_c}[1] = \frac{\mid I_c - \widetilde{I_c} \mid}{I_c} \tag{4.16}$$

$$F_{I_c} = \begin{cases} 0 & Si\ \widetilde{A}_{I_C} \leq K_e \\ T(\Delta_{I_c}[1]) & en\ otro\ caso \end{cases} \tag{4.17}$$

$$\Delta_{I_c}[2] = \frac{\mid \widetilde{I_c} - \widetilde{I_c}(T_c) \mid}{\widetilde{I_c}} \tag{4.18}$$

$$F_{I_c} = T(\Delta_{I_c}[2])) \tag{4.19}$$

Donde I_c es la corriente de colector a temperatura ambiente, T_a, e $I_c(T_c)$ es la corriente de colector medida a una temperatura superior, T_c.

Medida de distorsión

Como se ha indicado previamente, el valor *THD,* mostrado en la Ecuación 4.4, relaciona la potencia de los armónicos frente a la potencia de la componente fundamental, y por lo tanto permite medir cómo se separa el amplificador de un amplificador ideal, para una entrada sinusoidal pura.

Para la obtención del valor de THD se introducirá una señal sinusoidal pura de una amplitud que se puede ajustar para que, con la ganancia objetivo, dé una salida con una amplitud acorde al margen dinámico buscado. Esta señal de salida deberá tener una distorsión menor que un umbral. Con ello, la mejor adaptación se obtendría para una menor distorsión de señal de salida, que estaría indirectamente relacionada con un margen dinámico óptimo.

La sustitución de la medida del margen dinámico por la de distorsión podría dar lugar a que se consiguiera una baja distorsión debido a que la ganancia del circuito considerado fuera baja. Sin embargo, la función de adaptación dispone de un sumando que exige la ganancia buscada y que compensaría la adaptación en este caso. Con ello, un circuito con buena adaptación tendría una ganancia próxima a la buscada y también ofrecería una baja distorsión de salida para una señal de amplitud adecuada.

Para el cálculo de la amplitud de la señal sinusoidal de entrada se ha tenido en cuenta que el valor de la ganancia buscada según las condiciones de diseño consideradas, y también un factor de margen α que baje la exigencia. La división por 2 permite obtener la amplitud de la señal según se ve en la Ecuación 4.20. Finalmente, el sumando correspondiente a la medida de distorsión viene indicado en la Ecuación 4.22.

$$V_i(THD) = \frac{\alpha \cdot V_{cc}}{A_v \cdot 2} \tag{4.20}$$

$$\Delta_{THD} = \frac{\mid 0{,}01 - \widetilde{THD} \mid}{0{,}01} \tag{4.21}$$

$$F_{THD} = \begin{cases} 0 & Si\ \widetilde{THD} \leq 0{,}01 \\ T(\Delta_{THD}) & en\ otro\ caso \end{cases} \tag{4.22}$$

4.6.1. Funciones de transformación

Como se ha comentado previamente, los términos de la función de adaptación pueden utilizarse directamente o bien pueden transformarse mediante un juego de funciones

que tienen por objeto expandir o comprimir el rango de valores de dichos términos, y así aumentar o disminuir respectivamente la presión selectiva, lo que finalmente tendrá un efecto en la convergencia del algoritmo.

A continuación se indican las funciones consideradas, que se muestran gráficamente en la Figura 4.21.

1. Lineal

$$T(\Delta x) = \Delta x \tag{4.23}$$

2. Exponencial

$$T(\Delta x) = e^{\Delta x} - 1 \tag{4.24}$$

3. Logarítmica

$$T(\Delta x) = ln(1 + \Delta x) \tag{4.25}$$

4. Exponencial retrasada

$$T(\Delta x) = e^{\Delta x - 3} - e^{-3} \tag{4.26}$$

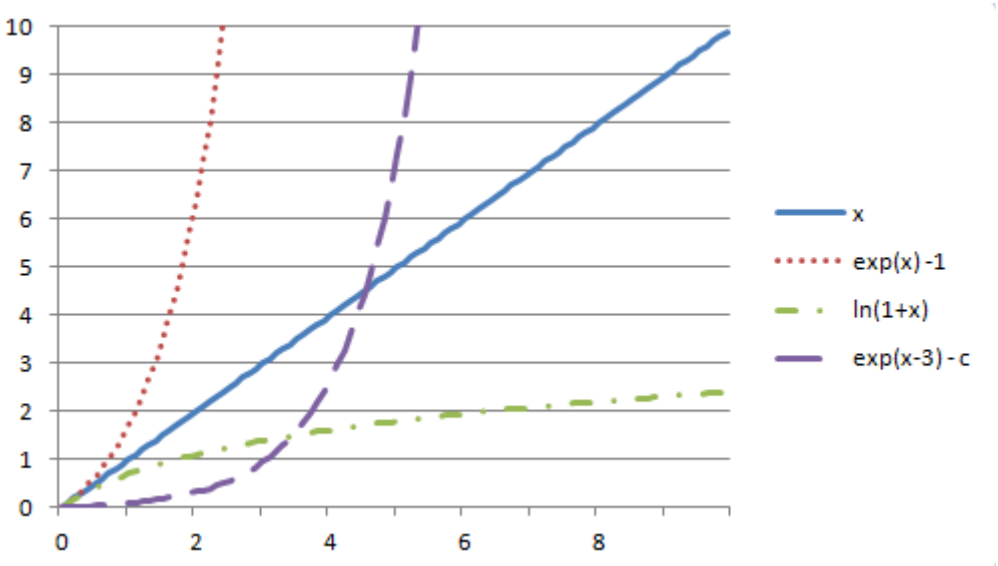


Figura 4.21. – Funciones de transformación

4.7. Embriones utilizados

En la Figura 4.22 se muestra la estructura de diferentes embriones que se utilizaran en los distintos experimentos realizados. El embrión 1 se compone de seis componentes y conexiones modificables, siendo los dos condensadores de desacoplo C_1 y C_6

componentes prefijados, en los que sólo puede evolucionar su valor. Las conexiones modificables Z_2, Z_3, Z_4 y Z_5 son completamente modificables, pudiendo evolucionar con cualquier expresión de transformación.

El embrión 2 se compone de cinco componentes y conexiones modificables, de nuevo, siendo los dos condensadores de desacoplo C_1 y C_5 componentes prefijados, en los que sólo puede evolucionar su valor. Las conexiones modificables Z_2, Z_3 y Z_4 son completamente modificables, pudiendo evolucionar con cualquier expresión de transformación.

(a) Embrión 1 **(b)** Embrión 2

Figura 4.22. – Embriones utilizados

Para conseguir que estos embriones contengan un componente de tipo fijo, como son los condensadores de desacoplo, mientras que se permite la evolución de su valor, es necesario realizar una modificación en la gramática utilizada que queda como se indica en Tabla 4.5. La modificación afecta a la regla de transformación de Expresion y se crea una nueva regla de transformación para una nueva variable llamada *CPB*.

4.8. Consideraciones sobre el margen dinámico

Como se ha comentado previamente hay algunas medidas que son muy fáciles de obtener para un humano, pero no son tan sencillas si se tratan de automatizar. En esta sección se describe la problemática encontrada que ha dado lugar al abandono de esta medida en favor de la medida de distorsión que resulta más sencilla de automatizar.

Como se ha indicado previamente, el margen dinámico corresponde a la máxima señal sinusoidal que puede entregar el amplificador sin saturar, y es muy fácil de obtener en un laboratorio utilizando un osciloscopio conectado a la salida del amplificador y un generador de señal a la entrada del mismo. Así, el margen dinámico vendrá dado por la excursión pico-pico de la señal de salida.

Para obtener esta medida de una forma automatizada, una primera aproximación consistiría en introducir una señal con una tensión suficientemente elevada para hacer saturar el amplificador y obtener como salida una señal recortada a los valores máximo y mínimo que puede ofrecer el amplificador, como se muestra en la Figura

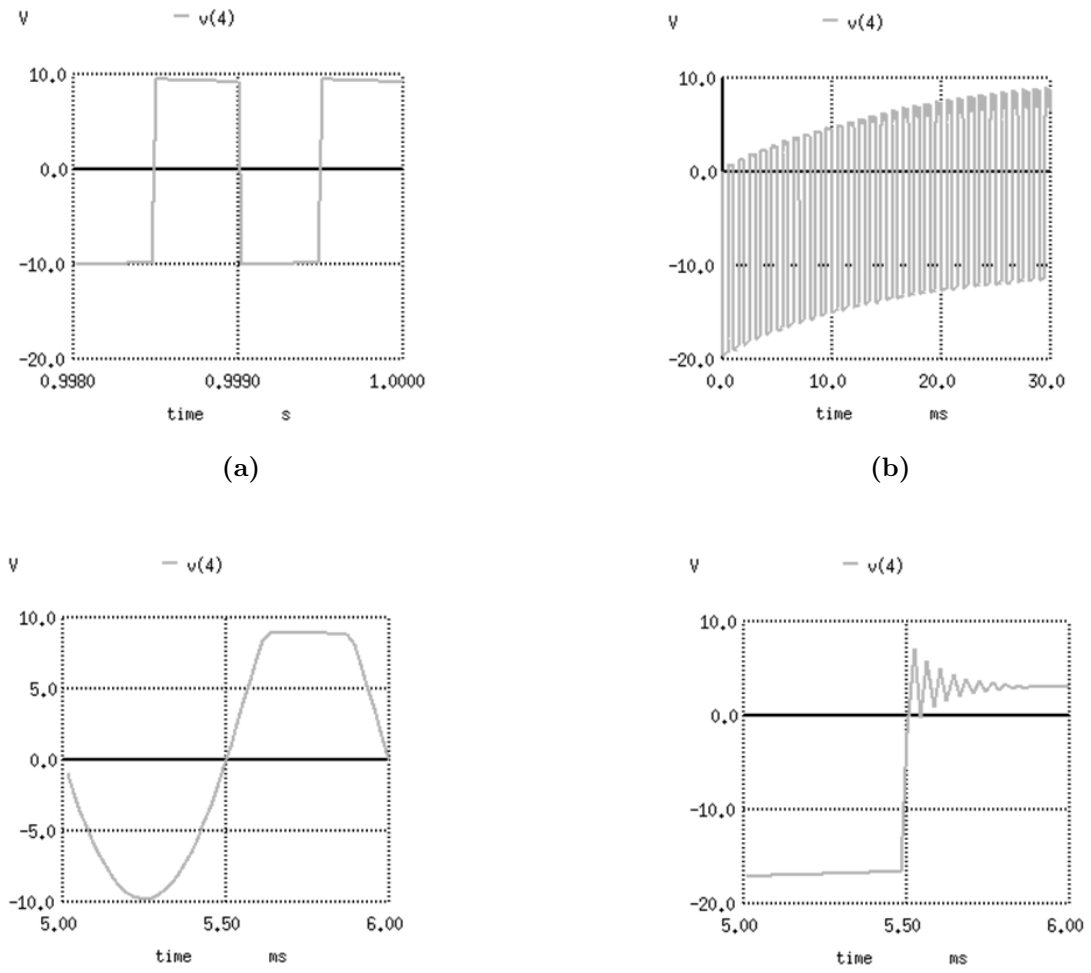
4.23a. De esta manera, sería suficiente con buscar los valores máximo y mínimo de la señal y su diferencia permitiría obtener el margen dinámico. Sin embargo, aparecen muchos casos problemáticos que se describen a continuación.

Un primer problema surge debido al régimen transitorio, que puede confundir al algoritmo, como se ve en la Figura 4.23b. Una solución sería esperar un tiempo para que el régimen transitorio termine y hacer la medición en régimen permanente. Sin embargo, esta solución tiene un impacto en tiempo de ejecución, pues es necesario simular un tiempo largo del circuito. Por ello, finalmente es necesario medir en los primeros ciclos de la señal, asumiendo esta situación. Se espera que que el mejor circuito obtenido por el algoritmo haya evolucionado de forma que se reduzca el efecto del transitorio en la medida.

Otro problema es debido a que los semiciclos de la señal pueden ser asimétricos, como se ve en la Figura 4.23c. Este problema se puede solucionar obteniendo el semiciclo que satura para un voltaje más bajo, que será el que determine el margen dinámico máximo.

Sin embargo, también hay otro tipo de casos como el mostrado en la Figura 4.23d, en el que aparece una oscilación en uno de los semiciclos. Este problema es más complejo, pues el valor máximo del semiciclo donde aparece la oscilación ofrece un margen dinámico más grande que el real. Se pueden valorar soluciones como hacer la medida de cada semiciclo, que funcionan en este caso, pero que ofrecen un mal resultado para salidas como la anteriormente comentada de la Figura 4.23c, pues un valor medio en cada semiciclo es lógicamente menor que el máximo o mínimo real, dando lugar a una medida más baja de la real.

El problema tal como está planteado está más cercano a un reconocimiento de patrones, lo que lo hace muy complejo de resolver de una forma sencilla. Por este motivo, y contando con la medida de distorsión, muy fácilmente automatizable, se abandonó la medida del margen dinámico.



(a) **(b)**

(c) **(d)**

Figura 4.23. – Salidas

```
S =                 Expresion
T =                 {'LIST','END', 'CUT', 'NOP', 'PARALLEL', 'SERIES',
                    'THREE_GROUND', 'PAIR_CONNECT', 'THREE_VCC', 'R',
                    'C','WIRE', 'e', '0', '1',...,'9',
                    '-12','-11',...,'-3'}
N =                 {RPB, CPB, FUN0, FUN1, FUN2, FUN3, CC,
                    ValorResistencia, ValorCondensador,
                    digito,digitoNoCero,expCondensador}
R = formado por las siguientes reglas de producción
Expresion =         'LIST', '(', CPB, ')', { '(', RPB, ')' }, '(', CPB,
                    ')';
RPB =               FUN0 | FUN1, '(', RPB, ')' | FUN2, '(', RPB, ')',
                    '(', RPB, ')' | FUN3, '(', RPB, ')', '(', RPB,')',
                    '(', RPB, ')' | CC, '(', RPB, ')';
CPB =               'C', ValorCondensador, '(', 'END', ')';
FUN0 =              'END' | 'CUT';
FUN1 =              'NOP' | 'WIRE';
FUN2 =              'PARALLEL';
FUN3 =              'SERIES' | 'THREE_GROUND' | 'PAIR_CONNECT' |
                    'THREE_VCC';
CC =                ( 'R', ValorResistencia, | 'C', ValorCondensador );
valorResistencia =  digitoNoCero, '.', digito, 'e', digito;
valorCondensador=   digitoNoCero, '.', digito, 'e', expCondensador;
digito =            '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8'
                    | '9';
digitoNoCero =      '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' |
                    '9';
expCondensador =    '-12' | '-11' | '-10' | '-9' | '-8' | '-7' | '-6' |
                    '-5' | '-4' | '-3';
```

Tabla 4.5. – Gramática modificada para el uso de condensadores de desacoplo

5. Arquitectura de la implementación

En este capítulo se describe el programa realizado para implementar la solución propuesta de diseño automatizado de amplificadores basada en el algoritmo evolutivo *Grammatical Evolution,* descrito en el *Capítulo 3,* conjuntamente con la aproximación propuesta, descrita en el Capítulo 4, basada en las expresiones de desarrollo, introducidas por Koza y sus colaboradores (Koza et al., 1999a).

El programa se ha desarrollado a medida en lenguaje *Java*. Otras herramientas utilizadas han sido *JavaCC* que es un generador de código para la implementación de *parsers*, que se ha utilizado para parsear las expresiones de desarrollo. Y para la simulación de circuitos se ha utilizado la herramienta *Ngspice*, que se describe con más detalle en la Sección 6.3.

El programa desarrollado se encuadra dentro del software de investigación o de desarrollo exploratorio, que se caracteriza por no tener todos los requisitos completamente definidos en una fase temprana del desarrollo, además de requerir una fácil adaptación a cambios que vienen propiciados por la obtención de resultados y que afectan a las especificaciones a lo largo de todo el desarrollo. Estas características han llevado a un desarrollo basado en un modelo de ciclo de vida iterativo o incremental, más próximo a las metodologías ágiles, con el objetivo de tener programas funcionales durante la mayor parte del ciclo, así como tolerar cambios en las especificaciones.

La complejidad de este tipo de programas ha motivado un uso extenso de pruebas automatizadas, basadas en la herramienta de pruebas unitarias *JUnit*, con objeto de asegurar el funcionamiento de las distintas clases conforme a las pruebas escritas. Adicionalmente se ha programado haciendo un uso extenso de comprobaciones, así como de aserciones, de forma que las condiciones de error se detecten lo antes posible, facilitando la depuración del programa.

La primera sección describe la arquitectura del programa realizado. Seguidamente se describen los paquetes principales mostrando los diagramas de clases que los componen. Después se muestra el diagrama de flujo de datos que describe las transformaciones realizadas sobre una cadena de bytes hasta obtener los valores de evaluación del circuito asociado. Finalmente se aborda la implementación de la paralelización del programa.

5.1. Arquitectura

En la Figura 5.1 se muestra una partición de primer nivel del programa mediante un diagrama de paquetes.

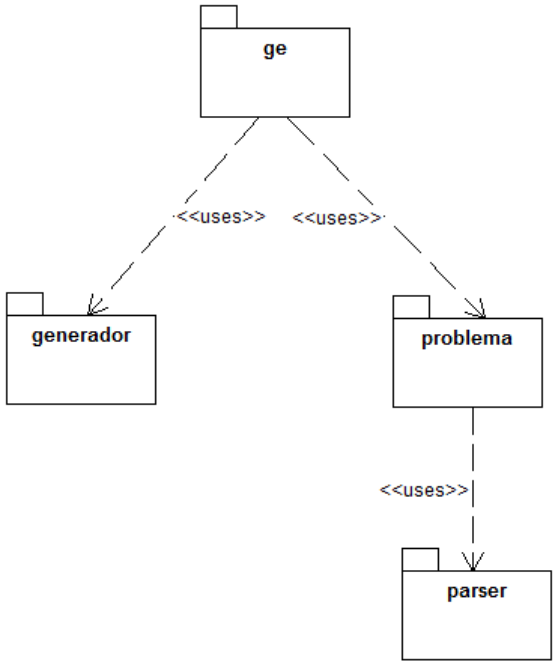


Figura 5.1. – Diagrama de paquetes

Donde los paquetes indicados son:

- *ge*: paquete que contiene las clases que implementan el algoritmo de *GE* como soporte de población, genotipo, operadores de cruce y mutación, etc...
- *generador*: contiene la funcionalidad de generación de expresiones a partir de una cadena binaria.
- *problema*: este paquete contiene la funcionalidad de evaluación de una solución que se encuentra como expresión. Utiliza el paquete parser para procesar la expresión recibida.
- *parser*: contiene la funcionalidad de parseado de la expresión generada y obtención de un árbol AST.

La arquitectura general del programa se ha definido de una forma abstracta de forma que se puedan implementar diferentes tipos de problemas. En este sentido, en una fase inicial del desarrollo se implementó un problema de regresión de expresiones simples, cuya gramática se ha mostrado en la Tabla 3.1. Por este motivo, en algunos diagramas de clases aparecerán referencias a las clases principales de este problema que se mantienen únicamente como referencia y, por lo tanto, no se van a describir en detalle.

5.2. Descripción de los paquetes

5.2.1. Paquete *ge*

El diagrama de clases del paquete *ge* es el siguiente:

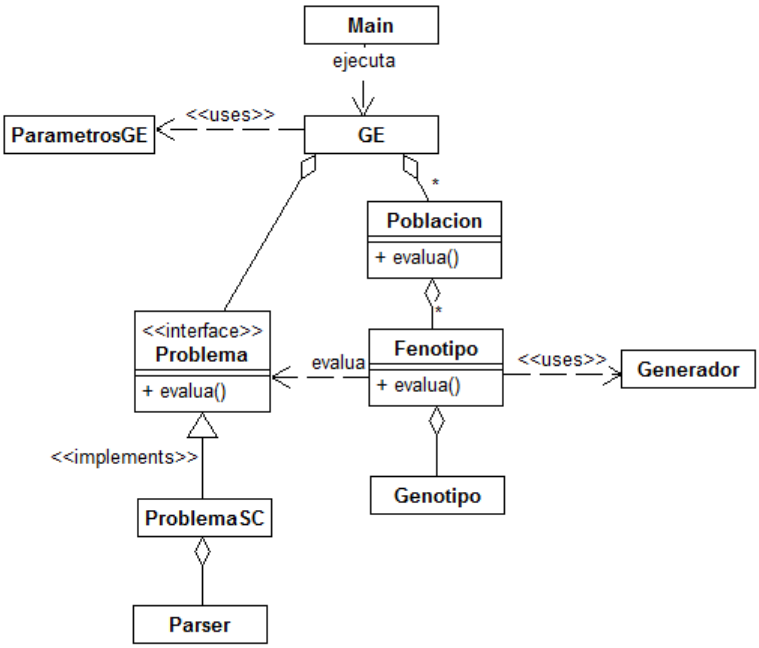


Figura 5.2. – Paquete ge

La clase principal ejecuta un algoritmo de *Grammatical Evolution* que se implementa en la clase *GE*. La configuración de parámetros del algoritmo, del problema a resolver y de los operadores utilizados, se realiza mediante el fichero *parametrosGE.properties* que se lee mediante la clase *ParametrosGE*.

La clase *GE* se compone de una población por cada generación del algoritmo. Las poblaciones se componen de una agregación de fenotipos, cada uno de los cuales contiene un genotipo. La clase Genotipo contiene una cadena de bytes, no mostrada en el diagrama al ser un tipo de datos primitivo de *Java*.

La clase *GE* también se compone de una instancia que implementa la interfaz *Problema*. El uso de una interfaz permite abstraer el algoritmo de la implementación concreta del problema que se va a resolver.

La superclase *Generador* realiza la función de decodificación de un genotipo y la obtención de una expresión. El uso de una superclase permite ocultar los detalles de la implementación concreta.

La evaluación de una población se implementa mediante la evaluación de todos los fenotipos que la componen. La evaluación de un fenotipo consiste en primer lugar en

la decodificación de su genotipo mediante un objeto de la clase *Generador* adecuado, que obtendrá una expresión de desarrollo o de regresión, según la clase concreta instanciada. A continuación se evalúa la evaluación de dicha expresión a través del método *evalua* de la la interfaz *problema* que recibe una expresión y devuelve un valor de adaptación.

5.2.2. Paquete *ge.operadores*

Este paquete contiene todos los operadores del *GE*, como son cruce, mutación y duplicación. También contiene la inicialización de cadenas binarias y la selección de padres y de supervivientes. Los diagramas de clases se muestran en la Figura 5.3a para los operadores que actúan sobre genotipos, y en la Figura 5.3b para los operadores que actúan sobre una población.

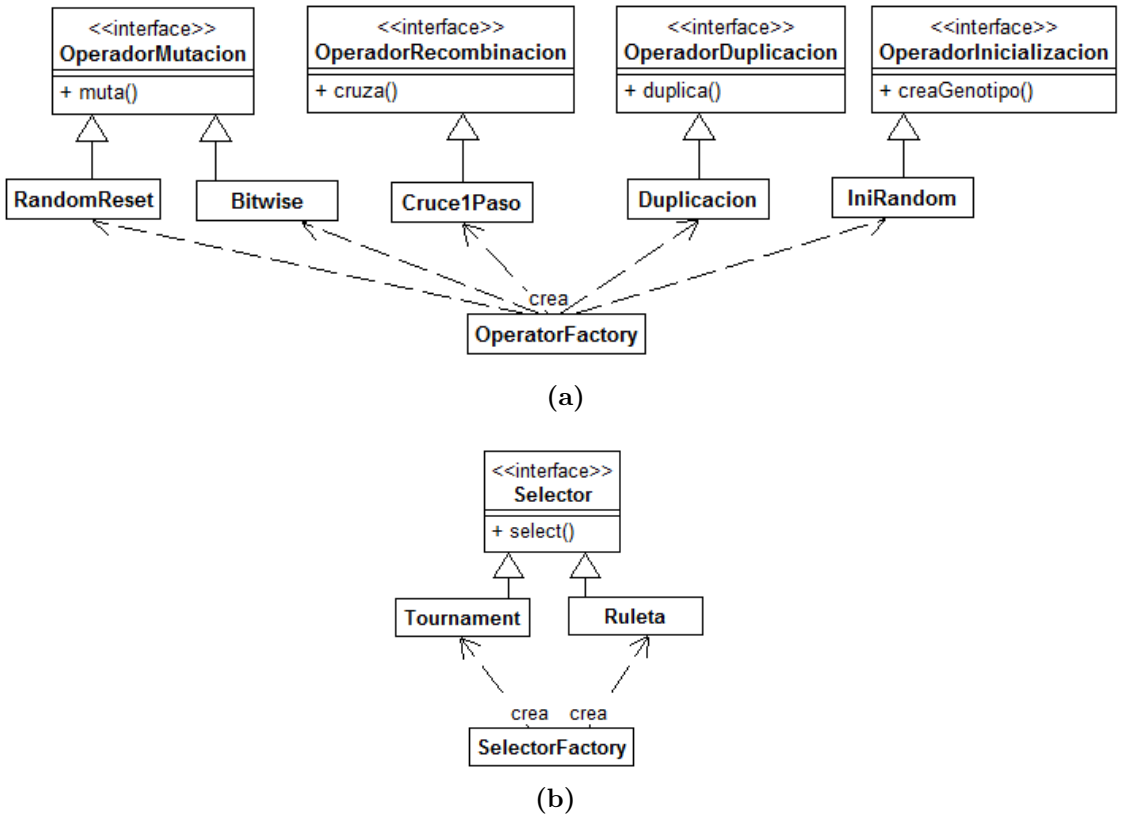


Figura 5.3. – Paquete *ge.operadores*: (a) sobre genotipos, (b) sobre población

En todos los casos se ha utilizado una interfaz para abstraer las operaciones de cada tipo de operador, de forma que se oculten los detalles de implementación. También se está utilizando un patrón de diseño *simple factory* (Freeman et al., 2004), que permite ocultar la creación de las clases concretas, que se seleccionan mediante parámetros indicados en un fichero de propiedades.

La interfaz *OperadorMutacion* define el método muta, que afecta a una instancia de la clase *Genotipo*. Se han desarrollado dos operadores concretos *RandomReset* y *Bitwise*, cuya implementación es acorde a la descrita en la Subsección 3.3.2.

La interfaz *OperadorRecombinacion* define el método *cruza*, que actúa sobre dos instancias de la clase *Genotipo* que son genotipos padre y obtiene dos nuevos genotipos hijos. Se ha desarrollado el operador concreto cruce1Paso cuya implementación es acorde a la descrita en la Subsección 3.3.1, con la modificación de limitar el tamaño de las cadenas de salida a un valor máximo definido en el fichero de parámetros, con objeto de limitar el efecto engorde, según se describe en la Sección 7.6.

La interfaz *OperadorDuplicacion* define el método duplica, que actúa sobre una instancia de la clase Genotipo. Se ha desarrollado la clase concreta *Duplicacion* con la funcionalidad indicada en Subsección 3.3.3.

La interfaz *OperadorInicializacion* define el método *creaGenotipo* que realiza la creación aleatoria de una instancia de la clase Genotipo. Se ha desarrollado la clase concreta *IniRandom* que genera una cadena aleatoria de tamaño aleatorio entre los límites indicados en el fichero de propiedades.

5.2.3. Paquete *generador*

Este paquete, cuyo diagrama de clases se muestra en la Figura 5.4, contiene la superclase abstracta *Generador*, la clase factoría *GeneradorFactory*, la excepción *GenException* y dos subpaquetes: *dev* y *exp*, que realizan la implementación de generadores de expresiones de desarrollo de circuitos y de expresiones simples respectivamente. La clase factoría permite la creación de un generador adecuado al tipo de problema seleccionado, ocultando la clase concreta de implementación.

La función de las clases hijas de *Generador* es la decodificación de una cadena de bytes, que representa un cromosoma, dando lugar a una expresión. La clase *GeneradorDev* genera expresiones de desarrollo de circuitos y la clase *GeneradorExp* genera expresiones simples. Centrándonos en generación de circuitos, la clase *GeneradorDev* crea un árbol de sintaxis abstracta, o AST, durante la decodificación de la cadena de bytes, que se implementa con las clases *Nodo*, *List* y RPB. Finalmente la clase *GenotipoVisitor*, que hace uso de la interfaz *Visitor*, implementa la generación de la expresión asociada al AST. Esta generación se implementa con el patrón de diseño *Visitor*.

El paquete *generador.dev* contiene la clase *TokenConst,* que usa el patrón de diseño *Singleton*, por lo que sólo hay una instancia de la misma. Esta instancia mantiene las cadenas de caracteres con los nombres de las funciones de transformación. También mantiene una relación de las funciones de transformación permitidas, que se puede configurar mediante un fichero de parámetros, con el cual se pueden activar o desactivar funciones de forma individual y así modificar la gramática de una forma sencilla, sin necesidad de recompilar de nuevo el programa.

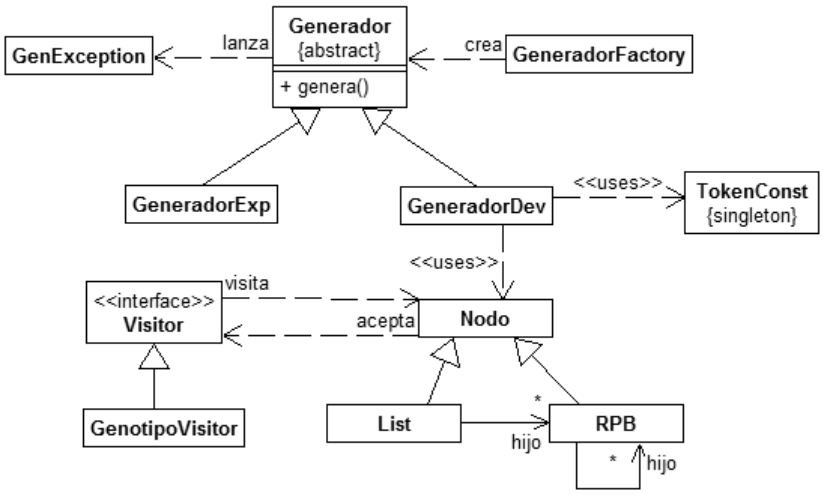


Figura 5.4. – Paquete generador

5.2.4. Paquete *parser* y *parser.dev*

La Figura 5.5 muestra las clases del paquete *parser*, que en sí no contiene clases comunes. Contiene dos subpaquetes *dev* y *exp* que contienen a su vez los parser de expresiones de desarrollo de circuitos y de expresiones simples respectivamente. Ambos *parsers* se han realizado utilizando la herramienta *JavaCC*, que permite la automatización de generación de *parsers* a partir de una gramática en formato EBNF anotada con código. El uso de dicha herramienta, así como los árboles de sintaxis abstracta (AST) resultado de cada parser, hacen que los dos paquetes no puedan abstraerse con una superclase común. En su lugar se ha optado por que cada implementación de problema sepa qué clase utilizar.

Se ha utilizado la funcionalidad de *jTree* implementada por *JavaCC*, de forma que se devuelve un AST, compuesto por clases generadas por la herramienta.

5.2.5. Paquete *problema*

Este paquete contiene la interfaz *Problema* y dos subpaquetes: *dev* y *exp*, que realizan la implementación de problemas de desarrollo de circuitos y de regresión de expresiones respectivamente. La Figura 5.6 muestra el diagrama de clases de este paquete y las clases principales de otros paquetes con las que se relacionan.

La superclase abstracta *ProblemaSC* permite implementar algunos métodos comunes a todas las implementaciones de problema. Las superclases abstractas *ProblemaDev* y *ProblemaExp* implementan los problemas de desarrollo de circuitos y de expresiones respectivamente, permitiendo implementar de nuevo, algunos métodos comunes a las clases finales de implementación, que se llaman *ProblemaAmp1*, ..., *ProblemaAmpN*, en el caso de desarrollo de amplificadores.

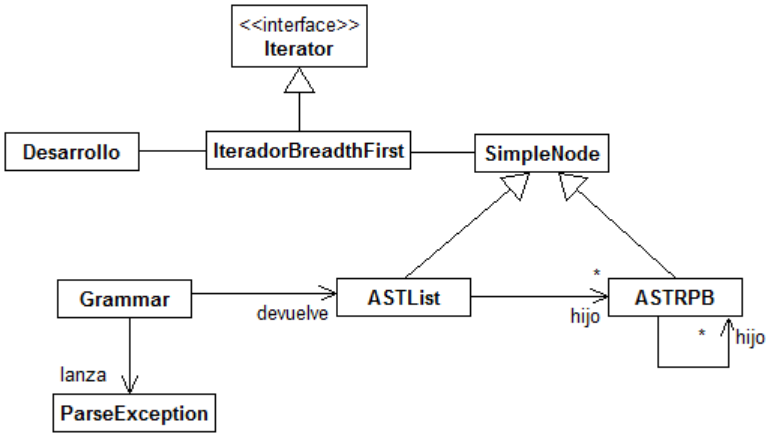


Figura 5.5. – Paquete parser

La clase *ProblemaDev* implementa el control de la evaluación de una expresión desde que se encuentra como una expresión, el desarrollo del circuito, la simulación del mismo y la obtención de los datos de salida de la misma. Este proceso se describe con mayor detalle en la Sección 5.3.

La clase *Parser*, perteneciente al paquete *parser.dev*, realiza el parseado de la expresión devolviendo un árbol de sintaxis abstracta o AST.

La clase *Desarrollo* junto con las clases *Fun* y *CC*, no mostradas en el diagrama, realizan la implementación del proceso de desarrollo y de las funciones de transformación. Este proceso recibe un AST y un embrión, al que aplica las transformaciones indicadas en el árbol.

La clase *NetList* pertenece al paquete *netlist* y realiza la función de obtención de una *netlist* a partir de un circuito representado mediante un grafo.

La clase *Spice* pertenece al paquete *spice* y realiza la función de ejecutar una simulación de circuitos mediante el programa *NgSpice*.

La clase *ProcesaSalidaSpice*, también perteneciente al paquete *spice*, procesa un fichero de salida del simulador y obtiene los valores necesarios para el cálculo de la adaptación que se realiza en la clase *ProblemaAmpN* correspondiente.

5.2.6. Paquete *grafo*

Este paquete contiene las clases que permiten representar un circuito como grafo. También se encuentra en este paquete la funcionalidad de simplificación y validación, así como la de transformación del circuito para la medida de la impedancia de salida. En la Figura 5.7 se muestra el diagrama de clases.

Un circuito se representa como un grafo compuesto de arcos y nodos, según la jerarquía de clases mostrada en la Figura 5.7a. Cada arco representa un componente

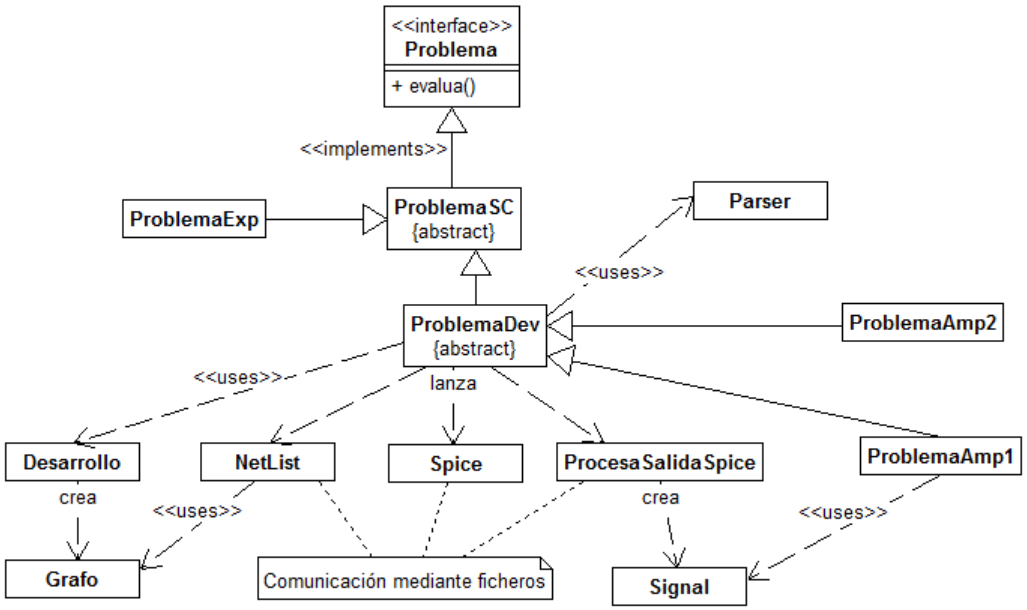


Figura 5.6. – Paquete problema

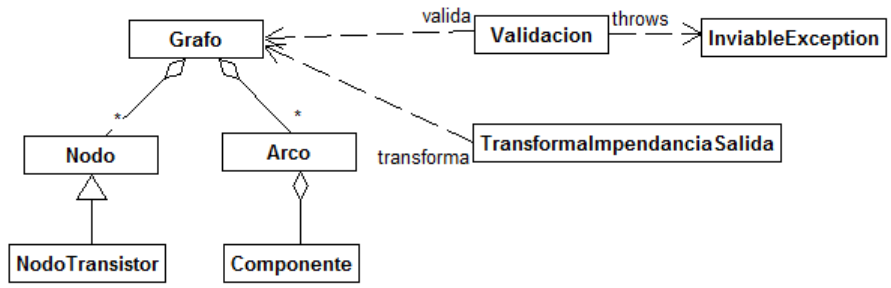
que puede ser un generador de señal, una fuente de alimentación, resistencias, condensadores y conexiones modificables. La representación de un transistor requiere un tratamiento especial al ser un componente con tres terminales. El modelado del mismo se hace con un tipo de nodo especial llamado nodo transistor, el cual dispone de tres arcos correspondientes a los tres terminales. Estos arcos también constituyen una excepción al tratamiento de arcos, cuando se genera la *netlist*, pues los tres arcos determinan un sólo componente. La clase *Componente* pertenece al paquete *netlist.*

La clase *Validación* implementa el algoritmo de simplificación y validación descrito en la Sección 4.3. En caso de que un circuito se determine inviable, lanzará la excepción *InviableException.*

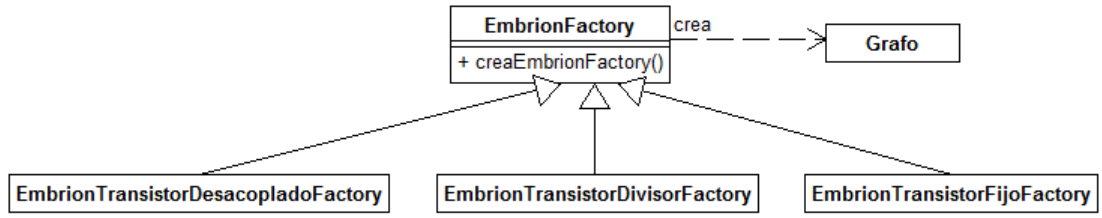
La clase *TransformaImpendanciaSalida* modifica la estructura fija de un circuito como se indica en la Sección 4.5.

Finalmente la creación de embriones se realiza mediante la superclase *EmbrionFactory* de forma que se independice el uso de la misma de las clases de implementación concretas. Estas clases son *EmbrionTransistorFijoFactory*, *EmbrionTransistorDivisorFactory* y *EmbrionTransistorDesacopladoFactory*, que corresponden a los embriones mostrados en la Figura 4.9b, la Figura 4.22a y la Figura 4.22b respectivamente.

Para parametrizar la creación de la factoría adecuada, se ha añadido la funcionalidad de factoría de factorías a la superclase *EmbrionFactory.*



(a)



(b)

Figura 5.7. – Paquete grafo

5.2.7. Paquete *netlist*

El paquete netlist contiene las clases necesarias para la conversión de un grafo en una netlist, o fichero que contiene los componentes del circuito y los análisis necesarios. Las clases que contiene son:

- *NetList*: esta clase realiza la funcionalidad de creación de una *netlist* a partir de un grafo dado.
- *Componente*: esta clase define un componente de circuito electrónico que se asocia a un arco del grafo. Esta clase aparecía mostrada en el diagrama de clases del paquete grafo, en la Figura 5.7a.

5.2.8. Paquete *spice*

Este paquete contiene las clases necesarias para lanzar una simulación utilizando el programa *Ngspice* y también para el proceso de un fichero de salida del simulador. Las clases que contiene son las siguientes:

- *Spice*: realiza la función de llamar a un programa externo *Ngspice* con los parámetros adecuados e indicando el fichero netlist que debe procesar. También realiza un control de casos error de este programa que producen un bucle infinito. Este control permite detener *Ngspice* cuando se detecta dicha condición, y se marca el circuito como inviable.

- *ProcesaSalidaSpice*: procesa un fichero de salida de *Ngspice* buscando el formato de volcado de los datos y realizando su agregación en varias clases *Signal*.
- *Signal*: contiene un conjunto de datos de una señal de voltaje o corriente a diferentes frecuencias. También puede contener valores de continua, constando de un sólo dato. Cada objeto *Signal* recibe un nombre que corresponde a si es tensión o corriente, y el nodo o rama donde se midió. Con este nombre será identificada por el método de evaluación de las clases *ProblemaAmpN* que obtienen el valor de adaptación final.

5.3. Diagrama de Flujo de Datos

En la Figura 5.8 se muestra un diagrama de flujo de datos que muestra las transformaciones sufridas desde la cadena de bytes (cromosoma) hasta el circuito final en formato netlist.

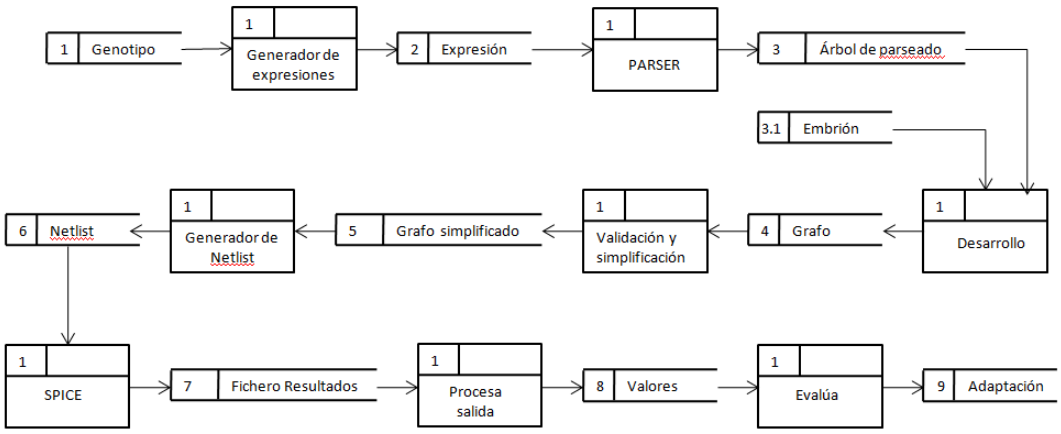


Figura 5.8. – DFD

El punto inicial de la transformación viene dado por una cadena binaria que se encuentra almacenada en una instancia de la clase Genotipo. La cadena binaria es un *array* de bytes. La secuencia de procesos se describe a continuación:

1. Proceso generador de expresiones, que se implementa mediante la clase *Generador*, decodifica la cadena y obtiene una expresión de desarrollo.
2. Proceso de parseado de la expresión, que se realiza mediante la clase *Parser*. La salida de la misma será un AST o árbol de sintaxis abstracta.
3. Proceso de desarrollo, que toma como entrada el árbol de parseado y también el embrión adecuado. La clase que implementa este proceso es la clase *Desarrollo*, cuya funcionalidad es realizar las transformaciones indicadas en la expresión de desarrollo sobre el embrión y obtener un embrión transformado que, junto

con la estructura fija, dará lugar al circuito final. Este circuito se almacena en el repositorio llamado *Grafo* y que es realmente una instancia de la clase del mismo nombre.

4. Proceso de validación y simplificación, descrito en la Sección 4.3, e implementado por la clase *Validacion*. Este proceso da lugar a un grafo simplificado.
5. Proceso de generación de un fichero netlist a partir del grafo. Este proceso se implementa en la clase *Netlist*.
6. Proceso de simulación del circuito electrónicos. Este proceso se realiza mediante el programa *Ngspice* y da lugar a un fichero de salida con los resultados de la simulación, que son los valores de voltajes y corrientes resultado de diferentes análisis.
7. Proceso del fichero de salida, que realiza la lectura del fichero y la obtención de los valores de voltajes y corrientes, bien de polarización, bien a diferentes frecuencias, temperatura, etc... estos valores se agregan en una composición de clases *Signal* que se pasa al proceso de evaluación.
8. Proceso de evaluación, que se implementa en una de las clases *ProblemaAmpN* y que calcula los resultados de ganancia, impedancias de entrada y salida, distorsión, frecuencias de corte, etc... con estos valores se calcula finalmente el valor de adaptación dependiendo del tipo de condiciones del experimento indicadas.

5.4. Implementación del paralelismo

En esta sección se describe la forma de implementación del paralelismo de este programa, comenzando por una justificación de la necesidad del mismo. Seguidamente se describe la librería utilizada para la comunicación entre nodos, que es Java RMI. Posteriormente se describe la arquitectura utilizada, así como la implementación de la misma.

5.4.1. Justificación del paralelismo

Se han realizado unas medidas de tiempos (*profiling*) sobre el programa para ver cómo se distribuye el consumo de tiempo entre los métodos del algoritmo. Para ello se ha utilizado la herramienta de *Java JVisualVM* que viene incluida con el JDK de *Java* a partir de la versión 6. Esta herramienta muestra el tiempo propio de un método (*self time*) por lo que no incluye el tiempo transcurrido en otros métodos llamados por el método medido. La monitorización afecta a la velocidad de ejecución, pero es una herramienta indispensable para averiguar en qué métodos se está consumiendo más tiempo de ejecución y poder optimizar la ejecución.

En la Figura 5.9 se muestra un diagrama de monitorización de tiempos (*profiling*) de la ejecución de un algoritmo *GE* con desarrollo de circuitos y evaluación de los mismos utilizando *NGSpice*. El diagrama se ha obtenido alrededor de la generación 35 y se observa que el tiempo se está consumiendo en el método *lanza* de la clase *Spice,* que corresponde a una ejecución de *NGSpice* para evaluar un circuito, representando más del 90 % del tiempo de ejecución. Con ello, casi la totalidad del tiempo de ejecución se consume en las evaluaciones de la adaptación de los circuitos mediante simulación, por lo que realizar una paralelización de las evaluaciones debería optimizar el algoritmo mediante el reparto de dicha carga entre varios nodos.

Hot Spots - Method	Self time [%]	Self time	Invocations
tfm.spice.Spice.**lanza** (String, String)		115139 ms (92.1%)	2054
java.lang.UNIXProcess$1$1.**run** ()		2556 ms (2%)	31
tfm.netlist.Netlist.**graba** (String)		758 ms (0.6%)	2054
tfm.netlist.Netlist.**valorToString** (double)		629 ms (0.5%)	18632
tfm.ge.operadores.MutacionBitwise.**muta** (t...		527 ms (0.4%)	8466
tfm.ge.random.RandomJava.**random** ()		471 ms (0.4%)	2619451
tfm.spice.ProcesaSalidaSpice.**lee** ()		409 ms (0.3%)	2054
tfm.util.Salida.**println** (String)		272 ms (0.2%)	22528
tfm.grafo.Grafo.**yaEsta** (tfm.grafo.Arco)		262 ms (0.2%)	1692070
tfm.grafo.Grafo.**check** ()		175 ms (0.1%)	64613
tfm.spice.ProcesaSalidaSpice.**readSigItem** (...		158 ms (0.1%)	29340
tfm.grafo.Grafo.**iteratorNodo** ()		143 ms (0.1%)	1730699
tfm.grafo.Grafo.**check** (tfm.grafo.Nodo)		141 ms (0.1%)	1692070

Figura 5.9. – *Profile* del programa

5.4.2. Descripción de *Java Remote Method Invocation*

Para la comunicación entre nodos y formar una arquitectura distribuida se han considerado varias opciones de herramientas entre las que se encuentran *Java RMI*, MPI, CORBA y *Web Services*. Finalmente se ha utilizado *Java RMI* que es una interfaz de *Java* que implementa comunicación entre objetos remotos, siendo una evolución del paradigma de *Remote Procedure Call* (RPC) a orientación a objetos. Con este modelo, es posible llamar desde una máquina virtual *Java*, en el mismo o diferente nodo, a un objeto dentro de otra máquina virtual.

En la Figura 5.10 se muestra el diagrama de clases básico de RMI, para implementación de un servicio remoto que se provee a través de la interfaz *Servicio*. Esta interfaz debe extender la interfaz *Remote* del paquete java.rmi, así como todos sus métodos, deben declararse indicando que lanzan la excepción *RemoteException* también del paquete java.rmi. Esta excepción notificará al cliente de que se ha producido una excepción en la ejecución del método remoto.

En el nodo servidor se instancia la clase *Servidor* que implementa la interfaz *Servicio*. Una vez se instancia, el objeto debe darse de alta con el nombre adecuado en el registro de *Java RMI*, implementado por la clase *Registry*. Dicho nombre permitirá

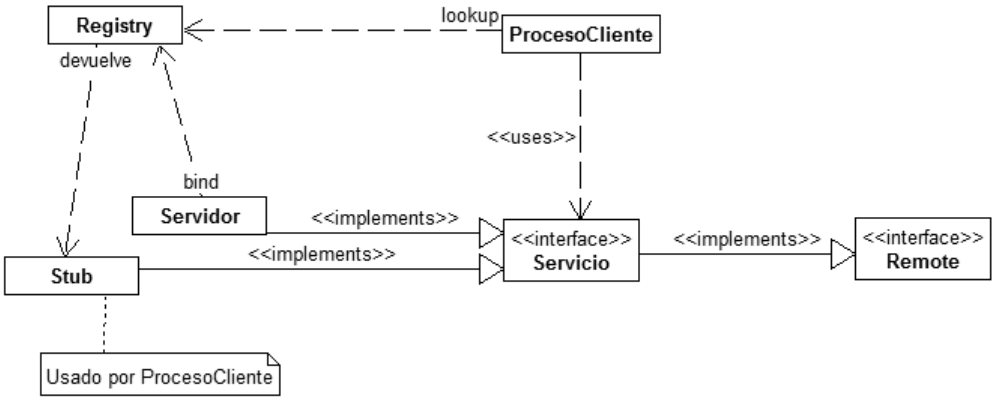


Figura 5.10. – Diagrama de clases básico de RMI

a los clientes localizar al objeto servidor y hacer uso del mismo. Una vez terminado el proceso de registro, el proceso de la máquina *Java* podría permanecer en espera, hasta que lleguen las peticiones de ésta o de otras máquinas *Java* diferentes.

En el nodo cliente, el programa que va a hacer uso del objeto remoto, debe localizar el registro de la máquina donde se encuentra el servidor. En cada nodo habrá un registro local que puede ser accedido desde otros nodos. Una vez accede al mismo, realiza una llamada al método *lookup* con el nombre del servidor que está buscando. Dicho método devuelve una referencia al objeto *Stub* que implementa la interfaz *Interfaz*. Dicho objeto *Stub* es implementado de forma automática por *Java RMI* y realiza las llamadas al nodo remoto.

Las llamadas del cliente al servidor son todas bloqueantes y esperan hasta la terminación de la llamada del método en la clase *Servidor*. *Java RMI* es inherentemente multihilo, por lo que será necesario utilizar las precauciones oportunas en la programación del código que utilice *Java RMI*.

5.4.3. Arquitectura del paralelismo

La partición del programa para su paralelización se ha realizado en un proceso maestro que implementa el algoritmo de *Grammatical Evolution*, hasta la decodificación de los genotipos para dar lugar a una expresión. la evaluación de dicha expresión se ha llevado a varios servidores remotos.

En la Figura 5.11 se muestra la comunicación entre nodos que se realiza a través de dos interfaces *ProblemaRemoto* y *Resultado*. La primera implementa la comunicación por la que se envía una expresión a un servidor remoto para su evaluación y la segunda es la interfaz de entrega de un resultado desde un servidor remoto al algoritmo principal. Estas interfaces se implementan por las clases *ProblemaServer* y *ResultadoServer* respectivamente.

La interfaz remota es la interfaz *Problema* que implementa el método *evalua* admitiendo como parámetro una expresión. Se ha paralelizado la ejecución de las evaluaciones de las diferentes expresiones y la simulación de los circuitos producidos. Una vez se ha realizado la interpretación de la expresión, se ha desarrollado el circuito a partir del embrión, se ha simplificado y validado, y finalmente simulado, se devuelve el valor de fitness al proceso maestro.

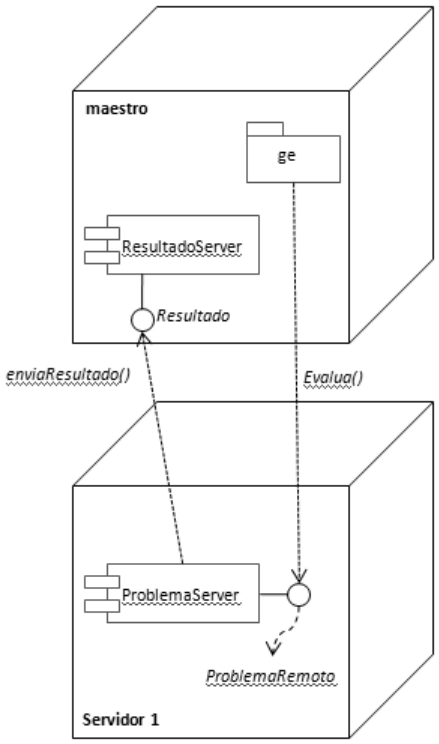


Figura 5.11. – Arquitectura del paralelismo

5.4.4. Paquete *rmi*

En la Figura 5.12 se muestra el diagrama de clases del paquete *rmi* que implementa el paralelismo en el programa y que se basa en dos interfaces: *ProblemaRemoto* y *Resultado*. El primero implementa la comunicación por la que se envía una expresión a un servidor remoto para su evaluación y el segundo la interfaz de entrega de un resultado desde un servidor remoto al algoritmo principal. La clase *ClientFactory* se ocupa de llamar a la clase *Registry,* no mostrada en el diagrama, para obtener los clientes *ResultadoStub* y *ProblemaRemotoStub*, el nombre de estas clases es ilustrativo, pues se crean de forma automática por la librería de RMI y no reciben nombre.

Un proceso servidor contiene una instancia de la clase ProblemaServer, que implementa un servidor remoto y que hereda de la clase *Thread,* con objeto de disponer

de un hilo de ejecución en el que realizar las evaluaciones. Esta clase se compone de una instancia de la interfaz *Problema*, que se implementa mediante una instancia hija *ProblemaAmp1, ProblemaAmp2, .., ProblemaAmpN,* con la que realiza la evaluación en sí. Esta clase recibe peticiones de evaluación desde el proceso maestro, que se encolan en una instancia de la clase *LinkedBlockingQueue,* propia de *Java*, que implementa una cola enlazada y protegida para uso en entorno multihilo. Esta cola permite bloquear el hilo de evaluación cuando la cola está vacía, esperando a que lleguen nuevas peticiones.

La clase *ProblemaGestorParalelo* se crea en el proceso maestro y mantiene una relación de todos los procesos servidores, junto con el número de expresiones enviadas a los mismos con objeto de realizar un balance de la carga. Estos contadores de carga se decrementarán cuando vayan llegando las evaluaciones de las expresiones enviadas.

La información de los servidores remotos se implementa mediante un *array* de clase *Server*, que contiene los datos de referencia de un servidor remoto y su carga actualizada.

5.4.5. Descripción del entorno utilizado

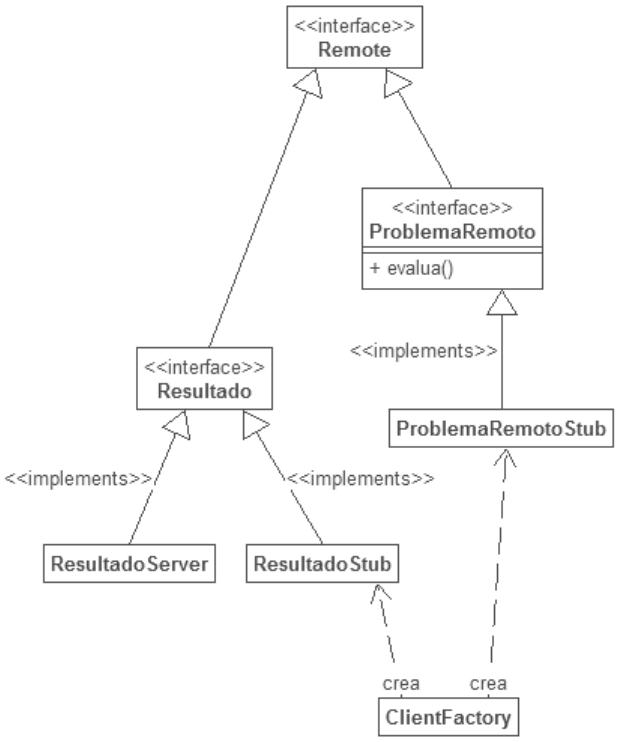
Se ha utilizado un cluster de cinco nodos que constan de un procesador de cuatro núcleos en cada nodo, $2GB$ de memoria principal y $250GB$ de disco. En la Tabla 5.1 se muestra la distribución del proceso maestro y los procesos servidores en los diferentes nodos del cluster utilizado. Los nodos se han instalado con Linux Ubuntu v12 y las versiones de las herramientas utilizadas se muestran en la Tabla 5.2.

Nodo	Servidores	Maestro
ce01	4	-
ce02	4	-
ce03	3	1
ce04	4	-
ce05	4	-

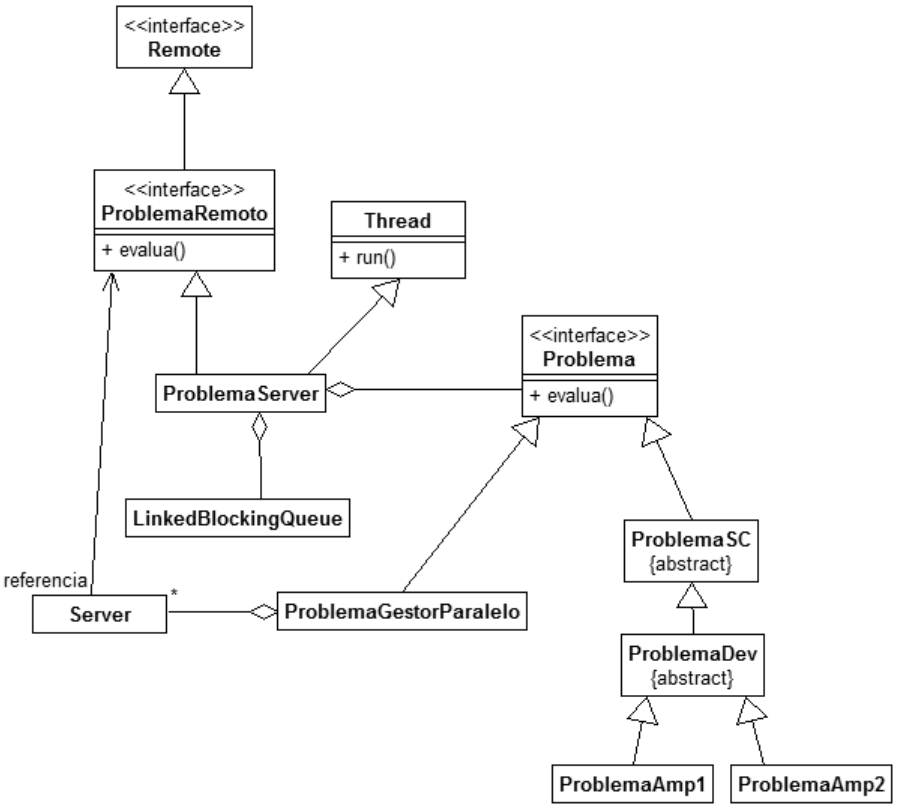
Tabla 5.1. – Distribución de los procesos en el cluster

JDK	6*update* 23
JavaCC	5,0
JUnit	4,8,1
Eclipse	Helios

Tabla 5.2. – Herramientas software



(a)



(b)

Figura 5.12. – Diagrama de clases del paquete *rmi*

6. Evaluación

En este capítulo se describen los experimentos realizados para la evaluación del sistema propuesto para el diseño automático de amplificadores de una etapa en configuración de emisor común con transistores BJT, y se muestran los resultados obtenidos. En primer lugar se indican las especificaciones de los dos amplificadores objetivo, junto con algunas consideraciones sobre las medidas. Posteriormente se describen los cuatro experimentos realizados, junto con los parámetros del algoritmo, las condiciones de diseño y los embriones utilizados. Seguidamente se describe el simulador de circuitos *Ngspice* utilizado en la evaluación, junto con los análisis realizados en dicha herramienta. A continuación se muestran los resultados de los experimentos, atendiendo a parámetros generalmente utilizados en la evaluación de algoritmos evolutivos y se describen los dos mejores circuitos obtenidos en cada experimento, mostrando las medidas obtenidas y se muestra el diagrama de Bode (respuesta en frecuencia) de la ganancia y de las impedancias de entrada y salida con la frecuencia. Finalmente se describen los circuitos diseñados manualmente y se calculan las mismas medidas obtenidas sobre los circuitos sintetizados por el algoritmo evolutivo.

6.1. Especificaciones

En la Tabla 6.1 se muestran dos juegos de especificaciones de dos amplificadores objetivo, que llamaremos condiciones de diseño 1 y condiciones de diseño 2. En ambos casos la estructura fija constará de una fuente de alimentación de $20V$, un generador de señal, una resistencia de entrada de 600Ω y una resistencia de carga de $100k\Omega$.

Las medidas de ganancia se realizarán para $f_1 = 1kHz$, $f_2 = 10kHz$ y $f_3 = 100kHz$ en las condiciones de diseño 1, mientras que sólo se harán a f_1 y f_2, con los mismos valores anteriores, en las condiciones de diseño 2. En ambas condiciones, la medida de las impedancias de entrada y salida se hará a la frecuencia $f_{media} = 10kHz$. Las medidas generales se harán a temperatura ambiente, $T_a = 27^oC$, y la medida de la corriente de colector a una temperatura más elevada, se hará a $T_c = 127^oC$. El valor del margen de corriente de colector será $K_e = 0{,}9$ que da lugar a un margen $1mA < I_C < 19mA$.

En relación con la medida de la distorsión, para obtener la amplitud de la señal de entrada se ha considerado un factor del 75 % en el caso de las condiciones de diseño

1, lo que ofrece un valor de amplitud de la señal de entrada de 1,9V. En el caso de las condiciones de diseño 2, es sabido a partir del diseño hecho por un diseñador humano, que el margen dinámico va a ser más estrecho que en el caso anterior. Por ello se ha considerado un factor del 18 %, lo que ofrece una amplitud de la señal de entrada de 7,17mV.

(a)

A_v	$12dB$
f_{low}	$10Hz$
Z_{in}	$20k\Omega$
Z_{out}	$6k\Omega$
Caída en $f_{low}/10$	$< 10dB/década$
THD	$< 1\,\%$
$I_C(T_a)$	$1mA < I_C < 19mA$
$I_C(T_c)$	$Ic(T_a)$

(b)

A_v	$48dB$
f_{low}	$1kHz$
Z_{in}	$2k\Omega$
Z_{out}	$6k\Omega$
Caída en $f_{low}/10$	$< 10dB/década$
THD	$< 1\,\%$
$I_C(T_a)$	$1mA < I_C < 19mA$
$I_C(T_c)$	$Ic(T_a)$

Tabla 6.1. – Especificaciones del amplificador: Condiciones de diseño 1 (a) Condiciones de diseño 2 (b)

6.2. Características de los experimentos realizados

En esta sección se describen los cuatro experimentos realizados, junto con los parámetros del algoritmo, las condiciones de diseño y los embriones utilizados. Se han realizado un conjunto de experimentos previos que han permitido elegir un conjunto de parámetros que, por ensayo y error, han dado lugar a buenos resultados. Con estos parámetros se han realizado cuatro experimentos finales que son los que se mostrarán en este trabajo. Los experimentos se realizan para las dos condiciones de diseño, indicadas en la Tabla 6.1, y para cada una de ellas se usan los dos embriones indicados en Figura 4.22, dando lugar a cuatro experimentos. En la Tabla 6.2 se muestran los parámetros generales del algoritmo evolutivo comunes a los cuatro experimentos realizados. En la Tabla 6.3 se muestran los parámetros específicos de cada experimento.

Debido a la naturaleza estocástica de los algoritmos evolutivos, cada experimento consta de 51 ejecuciones del algoritmo. El algoritmo agota el número de generaciones independientemente del valor de la mejor adaptación. El umbral de éxito definido se utilizará posteriormente para obtener una medida del número de éxitos, pero no se utilizará como condición de terminación.

Objetivo	Amplificador de una etapa en emisor común con un transistor BJT
Condiciones de diseño	Dependiente del experimento
Embrión	Dependiente del experimento
Representación	Cadena de codones (bytes) de longitud variable
Inicialización	Cadena de bytes aleatoria de longitud 1-10
Longitud máxima	250 bytes
Función de adaptación	Ecuación 4.5
Función de transformación	Logarítmica
Gramática	Tabla 4.5
Conjunto de funciones	Dependiente del experimento
Población	1000
Cruce	1 punto
Probabilidad de cruce	$0,5$
Mutación	Bitwise
Probabilidad de mutación	$0,001$
Selección de padres	Torneo de 3 miembros
Selección de reemplazo	Generacional
Elitismo	2
Wrapping	4
Umbral de éxito	$0,8$
Condición de terminación	300 generaciones
Repeticiones	51

Tabla 6.2. – Parámetros del algoritmo evolutivo basado en GE

6.3. Simulador de circuitos *Ngspice*

Para la evaluación de los circuitos se ha utilizado *Ngspice* (Nenzi and Vogt, 2011), que es una evolución del conocido simulador de circuitos SPICE3 (Nagel and Pederson, 1973).

Este simulador recibe como entrada un fichero, llamado netlist, que contiene la descripción del circuito que se va a simular. La netlist contiene todos los componentes que integran el circuito, indicando sus valores y los nodos a los que están conectados. Los nodos vienen determinados por un número de nodo, exigiendo que el nodo de masa venga identificado como cero. La netlist puede contener también los modelos de algunos componentes como los transistores. Finalmente, la netlist contiene también los análisis que se realizarán sobre el circuito con sus parámetros. Este simulador tiene dos modos de funcionamiento: un modo interactivo y un modo *batch.* El modo *batch* recibe la netlist como entrada y realiza los análisis indicados en la misma sobre el circuito descrito.

Los tipos de análisis que soporta *Ngspice* y que serán utilizados, son los siguientes:

1. .op, análisis de polarización, o también, *operation point*.
2. .ac, análisis de pequeña señal. Es un análisis en el que se utiliza una entrada suficientemente pequeña para considerar que el circuito se comporta de manera lineal.
3. .tran, análisis transitorio. Este análisis permite obtener la salida del amplificador para una entrada de voltaje mayor que el de gran señal. Permitirá hacer el estudio de la distorsión introducida por el circuito.
4. .four, trabaja sobre el análisis transitorio, realizando un análisis de Fourier que permite calcular la distorsión que introduce el circuito. Este análisis contempla 10 valores, siendo los dos primeros los valores de continua y la amplitud de la componente fundamental, lo que ofrece 8 valores para los armónicos. Finalmente también calcula el valor de *THD*, que se utilizará en el algoritmo exigiendo que sea menor del 1 %, que es un valor estándar en electrónica.

Para la obtención de las medidas anteriores se realizan tres ejecuciones de *Ngspice* por cada circuito tentativo.

1. Simulación a 27^oC. Se realiza un análisis de pequeña señal por décadas desde $1Hz$ hasta $100MHz$, obteniendo nueve medidas de la tensión en la resistencia de salida, la tensión posterior a la resistencia de entrada y la corriente proporcionada por la fuente de señal. Con estas medidas se obtiene la ganancia en las frecuencias indicadas en las condiciones de diseño, y la impedancia de entrada a la frecuencia de $10kHz$. También se obtienen la ganancia en la frecuencia de corte, así como una década inferior.
   Se realiza un análisis transitorio de $0{,}2ms$ con pasos de $2\mu s$, considerando una entrada de $10kHz$ y la amplitud dependiente de las condiciones de diseño, según se indica en la Sección 4.5. Posteriormente un análisis de Fourier de la señal de tensión de salida, que permitirá obtener el valor de THD.
2. Simulación a 27^oC. Este análisis se realiza sobre el circuito modificado para medir la impedancia de salida. Se realiza un análisis de polarización, para obtener la corriente de colector, y un análisis de pequeña señal a la frecuencia $10kHz$, donde se obtiene la tensión en la fuente de señal y la corriente proporcionada por la misma, con las que se obtiene la impedancia de salida.
3. Simulación a 127^oC. Se realiza un análisis de polarización, para obtener la corriente de colector a esta temperatura, que se comparará posteriormente con la medida a 27^oC, para comprobar la estabilidad de dicha corriente con la temperatura.

(a)

Experimento 1	
Condiciones de diseño	1 (Tabla 6.1a)
Embrión	1 (Figura 4.22a)
Conjunto de funciones	*R, C, WIRE, END, CUT, NOP, PARALLEL, SERIES*

(b)

Experimento 2	
Condiciones de diseño	1 (Tabla 6.1a)
Embrión	2 (Figura 4.22b)
Conjunto de funciones	*R, C, WIRE, END, CUT, NOP, PARALLEL, SERIES, THREE_GROUND, THREE_VCC, PAIR_CONNECT*

(c)

Experimento 3	
Condiciones de diseño	2 (Tabla 6.1b)
Embrión	1 (Figura 4.22a)
Conjunto de funciones	*R, C, WIRE, END, CUT, NOP, PARALLEL, SERIES*

(d)

Experimento 4	
Condiciones de diseño	2 (Tabla 6.1b)
Embrión	2 (Figura 4.22b)
Conjunto de funciones	*R, C, WIRE, END, CUT, NOP, PARALLEL, SERIES, THREE_GROUND, THREE_VCC, PAIR_CONNECT*

Tabla 6.3. – Parámetros específicos de cada experimento

6.4. Resultados

En la siguiente sección se muestran los resultados de los experimentos, atendiendo a parámetros generalmente utilizados en la evaluación de algoritmos evolutivos y se describen los dos mejores circuitos obtenidos en cada experimento, mostrando las medidas obtenidas y se muestra el diagrama de Bode de la ganancia y de las impedancias de entrada y salida.

En primer lugar se muestra una tabla resumen que incluye medidas habituales de evaluación de algoritmos evolutivos como *success rate* (*SR*) y *mean best fitness* (*MBF*). En la Tabla 6.4 se muestran los resultados de las 51 ejecuciones de cada algoritmo en cada experimento. Se muestra el número de éxitos, que corresponden al número de ejecuciones en los que la mejor adaptación es menor que el umbral fijado de $0,8$. Se muestra el *SR,* que corresponde al porcentaje de ejecuciones exitosas y el tiempo medio en minutos. En relación con los valores de mejor adaptación de cada ejecución, se muestran también los valores máximo y mínimo, el valor medio, que corresponde a la medida *MBF,* y la desviación típica.

Exp.	Cond. Diseño	Em-brión	Éxitos	SR	Tiempo medio	max BF	min BF	MBF	SD
1	1	1	8	$15,69\,\%$	$7,7'$	$2,057$	$0,669$	$1,181$	$0,285$
2	1	2	10	$19,61\,\%$	$7,6'$	$5,437$	$0,059$	$2,324$	$1,527$
3	2	1	35	$68,63\,\%$	$8,0'$	$1,500$	$0,065$	$0,616$	$0,280$
4	2	2	46	$90,20\,\%$	$6,9'$	$3,121$	$0,043$	$0,460$	$0,562$

Tabla 6.4. – Resumen de resultados

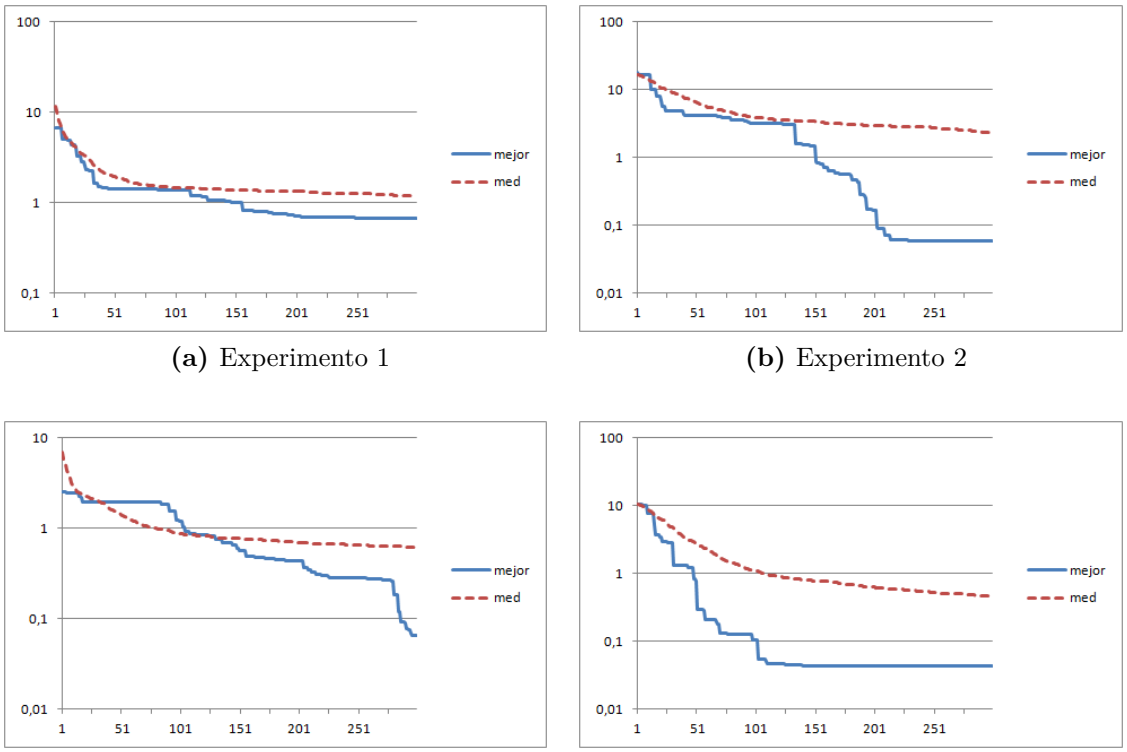
En la Tabla 6.5 se muestran valores estadísticos de la población final de cada experimento. Estos valores se obtienen en la última generación de cada ejecución, mostrándose los valores medios y desviaciones típicas de los mismos. A continuación se describe la obtención de dichos valores en cada ejecución. Los valores indicados como "longitud cadena" y "longitud expresada" se refieren al valor medio de las longitudes de las cadenas de bytes y las longitudes expresadas de las mismas de los individuos de la población correspondiente a la última generación. Ambos valores se miden en bytes. En el caso de longitud expresada se muestra además el valor medio más 3 veces la desviación típica, pues suponiendo una distribución normal dicho valor indica un límite superior para el 99,7 % de los casos. El valor *wrapping* corresponde a la media de veces que se produce el mecanismo de *wrapping* en los individuos de la población de la última generación. Este valor no tiene unidades. Los valores viables y expresables corresponden al número medio de genotipos viables y de genotipos expresables respectivamente que hay en la última generación. Estos valores tampoco tienen unidades.

Las curvas de convergencia del algoritmo se muestran en la Figura 6.1, donde aparece la evolución de la mejor adaptación con cada generación del algoritmo. Se muestra el

	longitud cadena		longitud expresada			wrapping		Expresables		Viables	
Exp.	med.	des.	med.	des.	$m+3d$	med.	des.	med.	des.	med.	des.
1	169, 80	50, 28	97, 00	28, 81	183, 41	0, 61	0, 77	793, 39	76, 19	780, 82	80, 75
2	166, 36	59, 44	98, 09	31, 63	192, 98	0, 82	1, 09	751, 39	79, 90	702, 55	88, 31
3	169, 12	56, 06	104, 53	23, 78	175, 86	0, 73	0, 87	761, 75	65, 30	748, 69	66, 53
4	194, 97	41, 44	99, 77	31, 45	194, 11	0, 31	0, 72	779, 86	71, 15	741, 27	76, 58

Tabla 6.5. – Estadísticas de la población final relacionadas con la longitud de los individuos, el mecanismo de wrapping y el número de individuos expresables/viables

valor medio de las mejores adaptaciones de las 51 ejecuciones y también la evolución de la mejor ejecución para cada experimento.



(a) Experimento 1 **(b)** Experimento 2

(c) Experimento 3 **(d)** Experimento 4

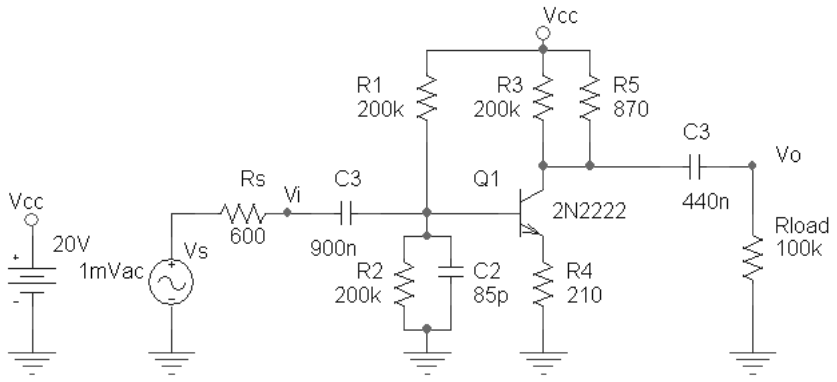
Figura 6.1. – Convergencia del algoritmo

A continuación se muestran los mejores circuitos obtenidos en cada experimento. Los circuitos se muestran sin realizar ninguna simplificación adicional a las que van integradas en el propio algoritmo, según se indicaba en la Sección 4.3. Esto se hace con objeto de ver los resultados directos del algoritmo, sin embargo, para una aplicación real sería necesario realizar simplificaciones de componentes en serie o paralelo.

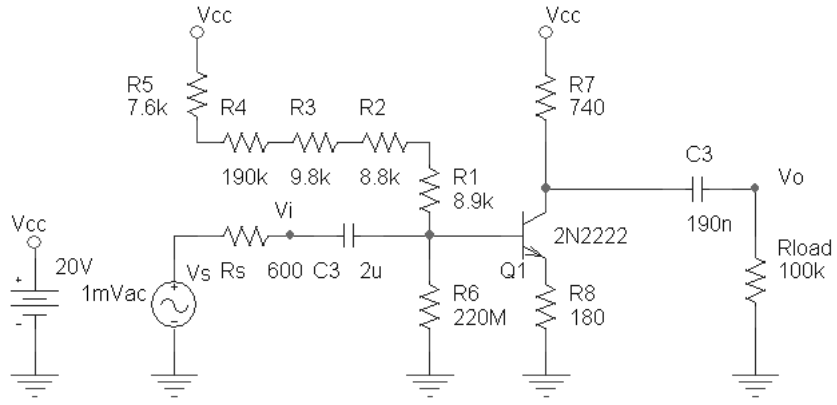
6.4.1. Experimento 1

El experimento 1 tiene como objetivo conseguir un amplificador de una etapa con un transistor bipolar en configuración de emisor común que cumpla las especificaciones de diseño indicadas en la Tabla 6.1a, utilizando el circuito embrión indicado en la Figura 4.22a.

Se muestran los dos mejores circuitos, etiquetados como circuito 1 y circuito 2 que corresponden al mejor circuito obtenido y al segundo mejor respectivamente. El circuito 1 se muestra en la Figura 6.2a y en la Tabla 6.6a se muestran las medidas obtenidas de la simulación del mismo. El circuito 2 se muestra en la Figura 6.2b y en la Tabla 6.6b se muestran las medidas obtenidas de la simulación del mismo. Finalmente en la Figura 6.3 se muestran las curvas de de ganancia y de las impedancias de entrada y salida con la frecuencia de ambos circuitos.



(a) Circuito 1



(b) Circuito 2

Figura 6.2. – Circuitos obtenidos en el experimento 1

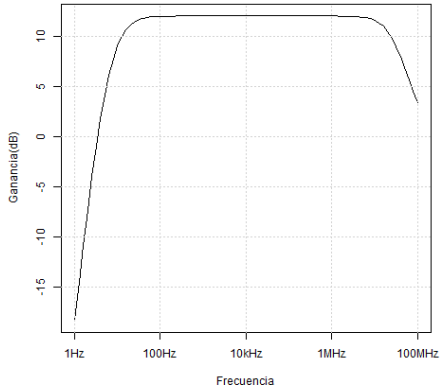
(a) Circuito 1

$A_v(dB)$ a f_1,f_2y f_3	$12,00$	$12,00$	$12,00$
$Z_{in}(k\Omega)$ a f_{media}	$20,00$		
$Z_{out}(k\Omega)$ a f_{media}	$0,86$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$9,00$	$-18,33$	
THD	$0,32$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$9,72$	$9,23$	$94,96\,\%$
$Fitness$	$0,669$		

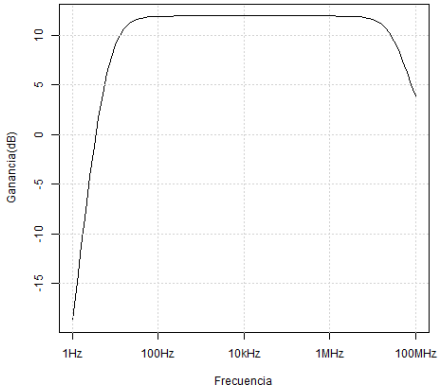
(b) Circuito 2

$A_v(dB)$ a f_1,f_2y f_3	$11,97$	$11,97$	$11,97$
$Z_{in}(k\Omega)$ a f_{media}	$20,00$		
$Z_{out}(k\Omega)$ a f_{media}	$0,74$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$9,05$	$-18,75$	
THD	$0,54$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$1,04$	$1,00$	$96,15\,\%$
$Fitness$	$0,685$		

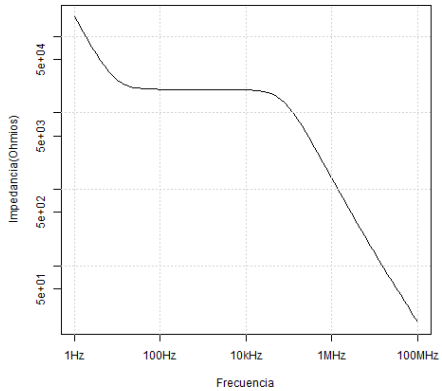
Tabla 6.6. – Medidas de los circuitos del experimento 1



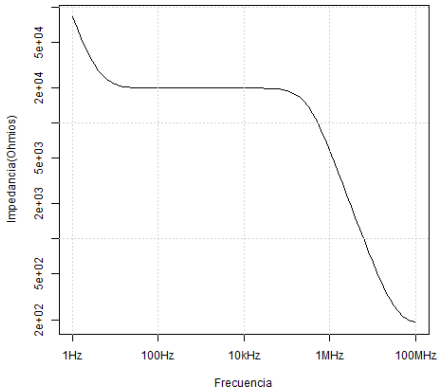
(a) A_v circuito 1



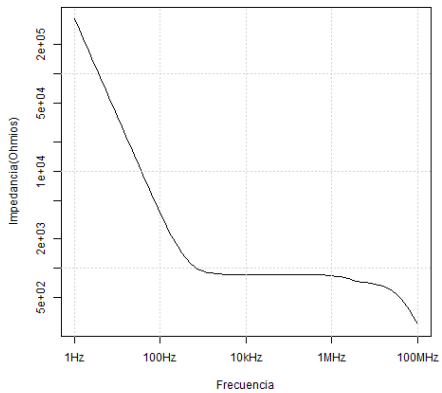
(b) A_v circuito 2



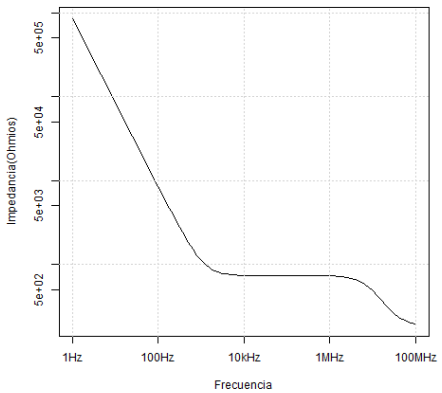
(c) Z_{in} circuito 1



(d) Z_{in} circuito 2



(e) Z_{out} circuito 1



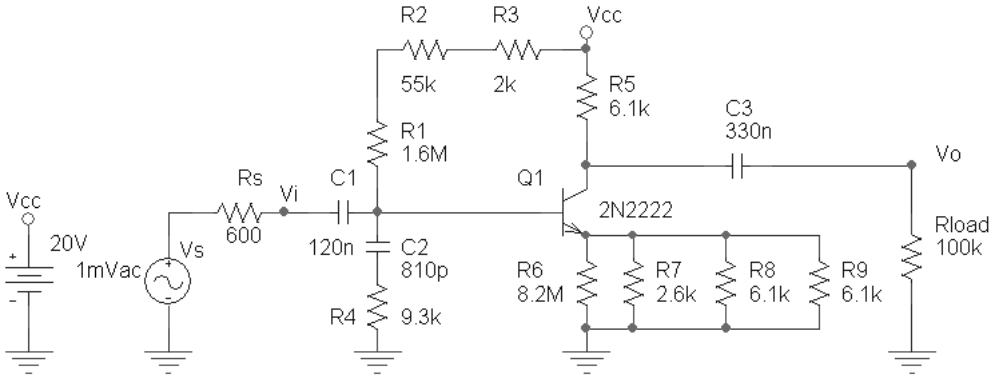
(f) Z_{out} circuito 2

Figura 6.3. – Diagramas de Bode del experimento 1

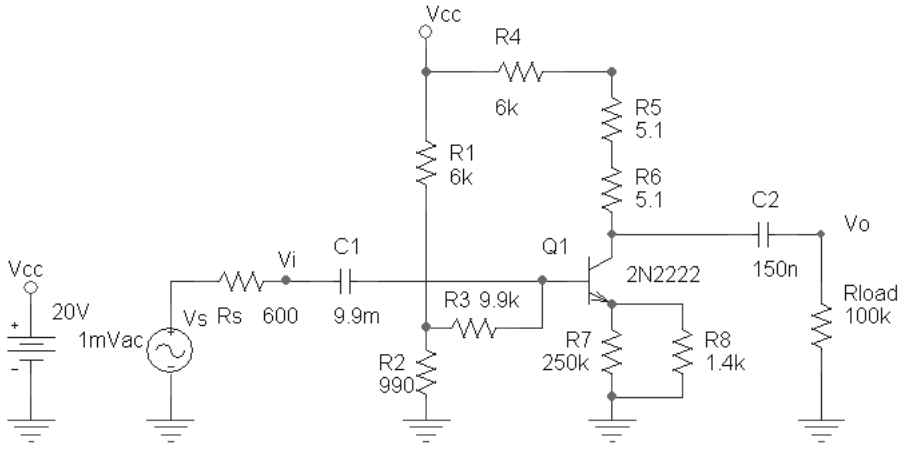
6.4.2. Experimento 2

El experimento 2 tiene como objetivo conseguir un amplificador de una etapa con un transistor bipolar en configuración de emisor común que cumpla las especificaciones de diseño indicadas en la Tabla 6.1a, utilizando el circuito embrión indicado en la Figura 4.22b.

Se muestran los dos mejores circuitos, etiquetados como circuito 3 y circuito 4 que corresponden al mejor circuito obtenido y al segundo mejor respectivamente. El circuito 3 se muestra en la Figura 6.4a y en la Tabla 6.7a se muestran las medidas obtenidas de la simulación del mismo. El circuito 4 se muestra en la Figura 6.4b y en la Tabla 6.7b se muestran las medidas obtenidas de la simulación del mismo. Finalmente en la Figura 6.5 se muestran las curvas de de ganancia y de las impedancias de entrada y salida con la frecuencia de ambos circuitos.



(a) Circuito 3



(b) Circuito 4

Figura 6.4. – Circuitos obtenidos en el experimento 2

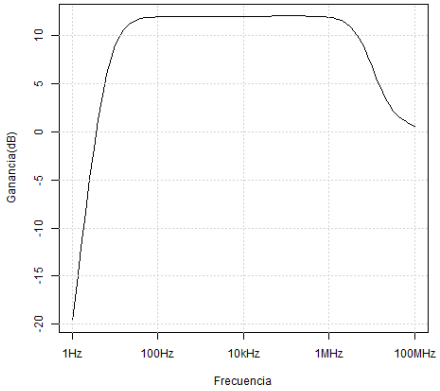
(a) Circuito 3

$A_v(dB)$ a f_1,f_2y f_3	$11,99$	$12,00$	$12,04$
$Z_{in}(k\Omega)$ a f_{media}	$20,01$		
$Z_{out}(k\Omega)$ a f_{media}	$6,09$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$9,00$	$-19,60$	
THD	$0,64$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$1,38$	$1,32$	$95,65\,\%$
$Fitness$	$0,059$		

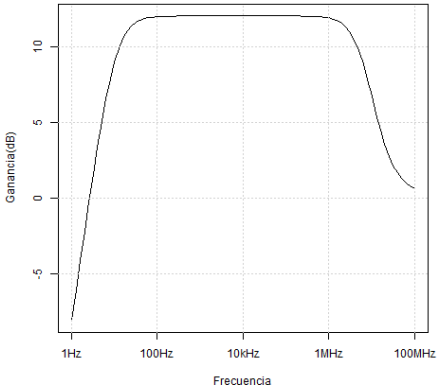
(b) Circuito 4

$A_v(dB)$ a f_1,f_2y f_3	$12,00$	$12,00$	$12,00$
$Z_{in}(k\Omega)$ a f_{media}	$10,14$		
$Z_{out}(k\Omega)$ a f_{media}	$6,00$		
$A_v(f_l)(dB)$ y $A_v(f_l/10)(dB)$	$8,99$	$-8,05$	
THD	$0,43$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$1,47$	$1,32$	$89,80\,\%$
$Fitness$	$0,498$		

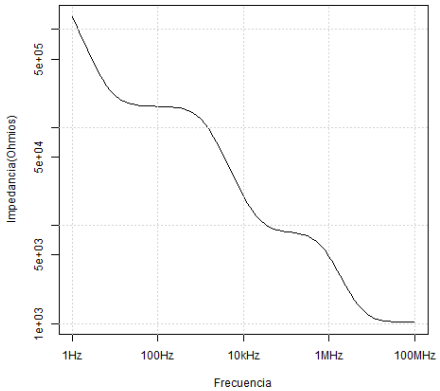
Tabla 6.7. – Medidas de los circuitos del experimento 2



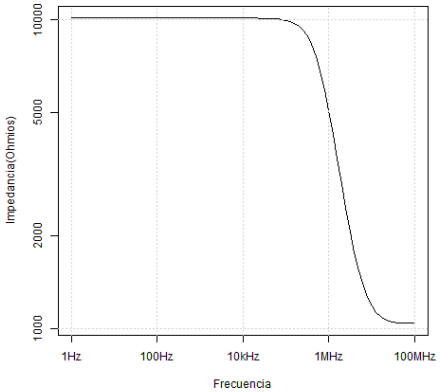
(a) A_v circuito 3



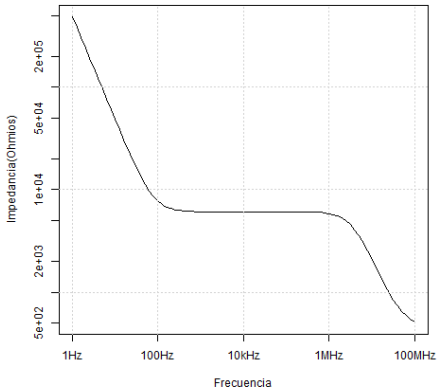
(b) A_v circuito 4



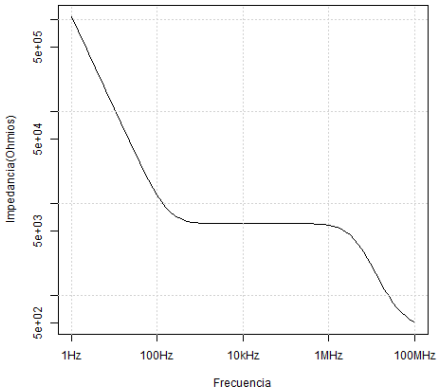
(c) Z_{in} circuito 3



(d) Z_{in} circuito 4



(e) Z_{out} circuito 3



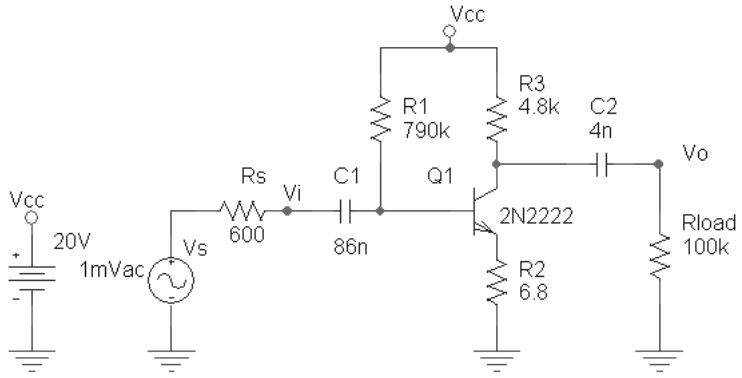
(f) Z_{out} circuito 4

Figura 6.5. – Diagramas de Bode del experimento 2

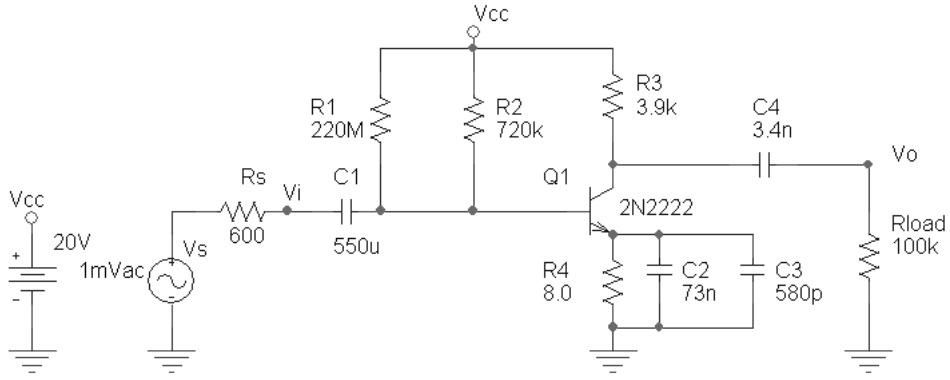
6.4.3. Experimento 3

El experimento 3 tiene como objetivo conseguir un amplificador de una etapa con un transistor bipolar en configuración de emisor común que cumpla las especificaciones de diseño indicadas en la Tabla 6.1b, utilizando el circuito embrión indicado en la Figura 4.22a.

Se muestran los dos mejores circuitos, etiquetados como circuito 5 y circuito 6 que corresponden al mejor circuito obtenido y al segundo mejor respectivamente. El circuito 5 se muestra en la Figura 6.6a y en la Tabla 6.8a se muestran las medidas obtenidas de la simulación del mismo. El circuito 6 se muestra en la Figura 6.6b y en la Tabla 6.8b se muestran las medidas obtenidas de la simulación del mismo. Finalmente en la Figura 6.7 se muestran las curvas de de ganancia y de las impedancias de entrada y salida con la frecuencia de ambos circuitos.



(a) Circuito 5



(b) Circuito 6

Figura 6.6. – Circuitos obtenidos en el experimento 3

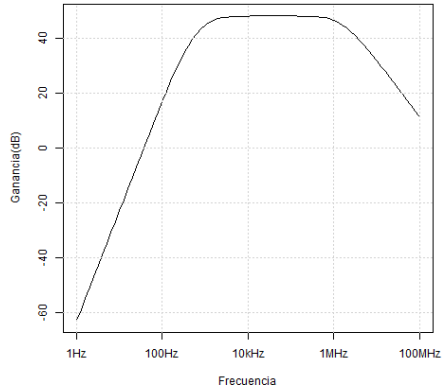
(a) Circuito 5

$A_v(dB)$ a f_1 y f_2	$48,21$	$48,23$	
$Z_{in}(k\Omega)$ a f_{media}	$1,98$		
$Z_{out}(k\Omega)$ a f_{media}	$5,99$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$44,99$	$16,98$	
THD	$0,98$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$3,15$	$3,01$	$95,56\,\%$
$Fitness$	$0,065$		

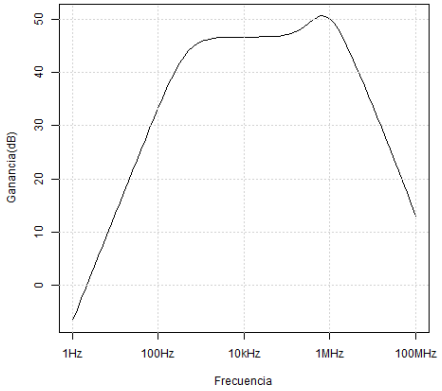
(b) Circuito 6

$A_v(dB)$ a f_1 y f_2	$46,64$	$47,07$	
$Z_{in}(k\Omega)$ a f_{media}	$2,00$		
$Z_{out}(k\Omega)$ a f_{media}	$6,01$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$45,84$	$33,34$	
THD	$0,61$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$3,52$	$3,37$	$95,74\,\%$
$Fitness$	$0,113$		

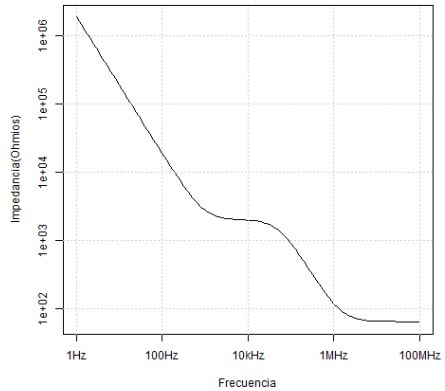
Tabla 6.8. – Medidas de los circuitos del experimento 3



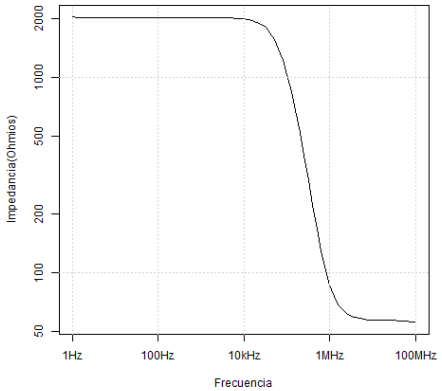
(a) A_v circuito 5



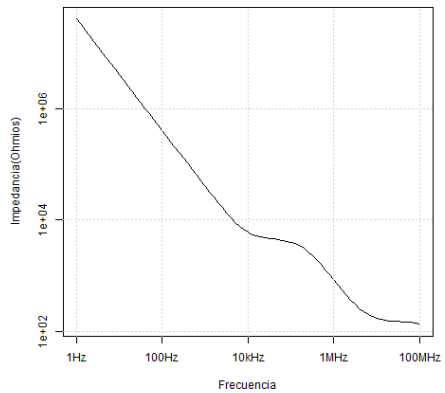
(b) A_v circuito 6



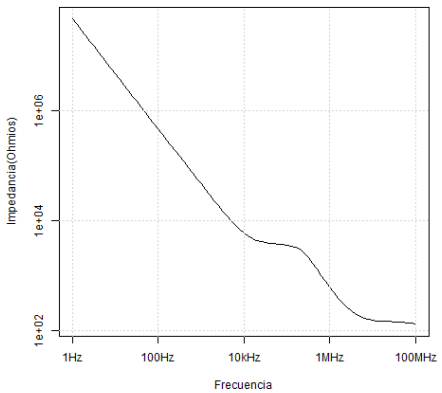
(c) Z_{in} circuito 5



(d) Z_{in} circuito 6



(e) Z_{out} circuito 5



(f) Z_{out} circuito 6

Figura 6.7. – Diagramas de Bode del experimento 3

6.4.4. Experimento 4

El experimento 4 tiene como objetivo conseguir un amplificador de una etapa con un transistor bipolar en configuración de emisor común que cumpla las especificaciones de diseño indicadas en la Tabla 6.1b, utilizando el circuito embrión indicado en la Figura 4.22b.

Se muestran los dos mejores circuitos, etiquetados como circuito 7 y circuito 8 que corresponden al mejor circuito obtenido y al segundo mejor respectivamente. El circuito 7 se muestra en la Figura 6.8a y en la Tabla 6.9a se muestran las medidas obtenidas de la simulación del mismo. El circuito 8 se muestra en la Figura 6.8b y en la Tabla 6.9b se muestran las medidas obtenidas de la simulación del mismo. Finalmente en la Figura 6.9 se muestran las curvas de de ganancia y de las impedancias de entrada y salida con la frecuencia de ambos circuitos.

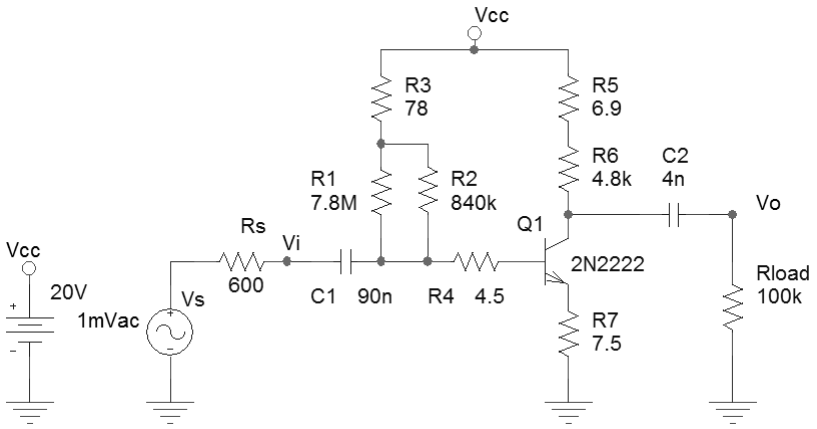
(a) Circuito 7

$A_v(dB)$ a f_1 y f_2	$47,99$	$48,01$	
$Z_{in}(k\Omega)$ a f_{media}	$2,00$		
$Z_{out}(k\Omega)$ a f_{media}	$6,00$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$45,00$	$17,27$	
THD	$0,93$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$3,27$	$3,12$	$95,41\,\%$
$Fitness$	$0,043$		

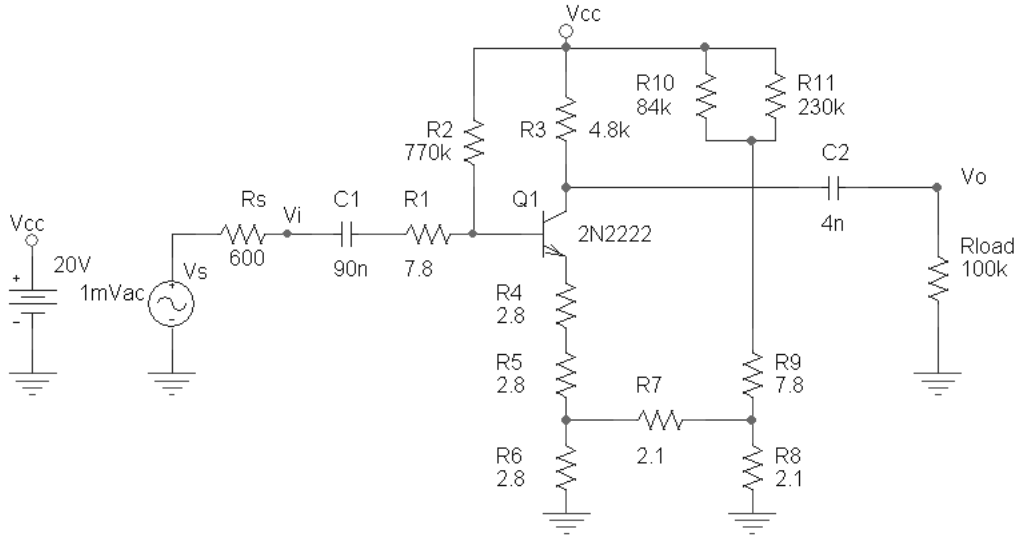
(b) Circuito 8

$A_v(dB)$ a f_1 y f_2	$48,03$	$48,04$	
$Z_{in}(k\Omega)$ a f_{media}	$2,00$		
$Z_{out}(k\Omega)$ a f_{media}	$6,00$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$45,04$	$17,30$	
THD	$0,93$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$3,22$	$3,08$	$95,65\,\%$
$Fitness$	$0,046$		

Tabla 6.9. – Medidas de los circuitos del experimento 4

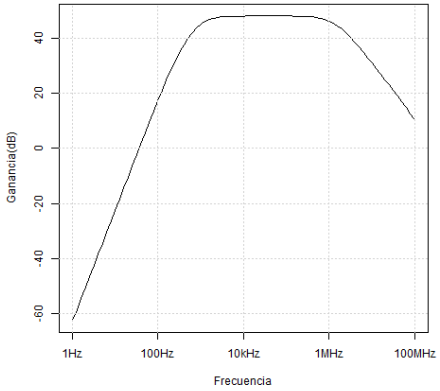


(a) Circuito 7

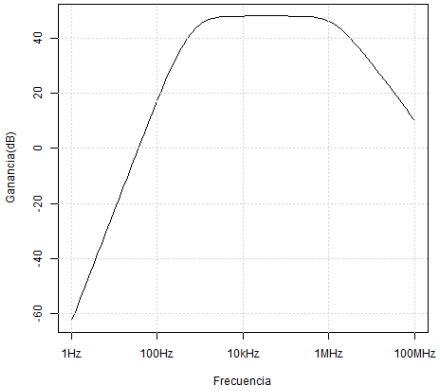


(b) Circuito 8

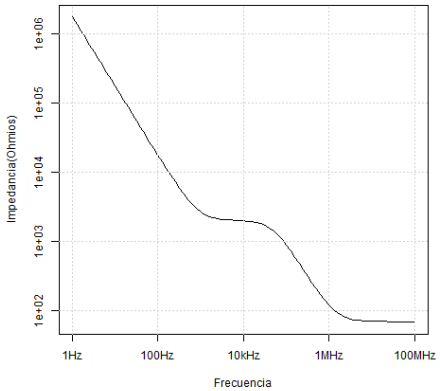
Figura 6.8. – Circuitos obtenidos en el experimento 4



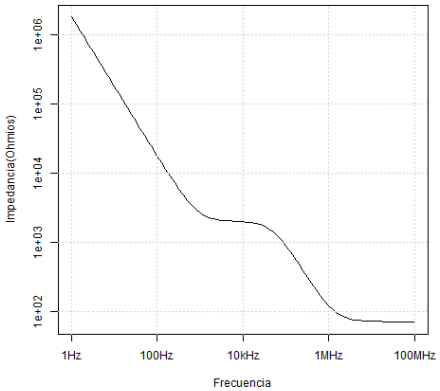
(a) A_v circuito 7



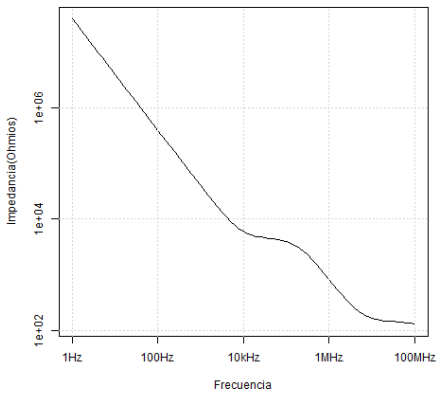
(b) A_v circuito 8



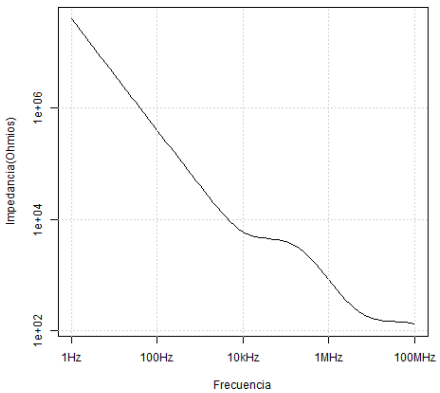
(c) Z_{in} circuito 7



(d) Z_{in} circuito 8



(e) Z_{out} circuito 7

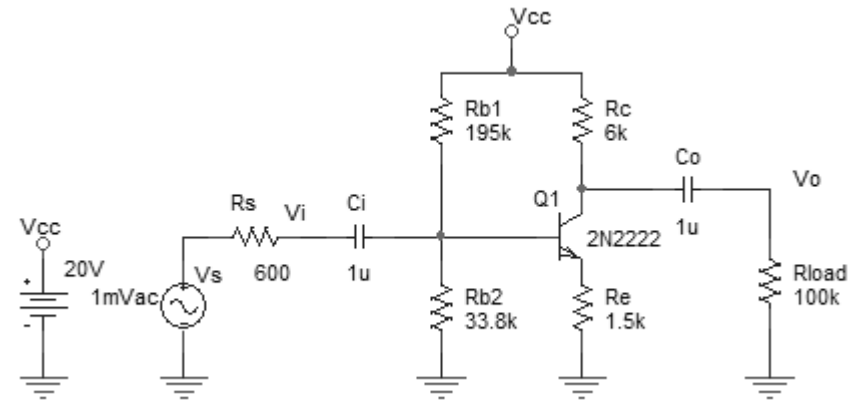


(f) Z_{out} circuito 8

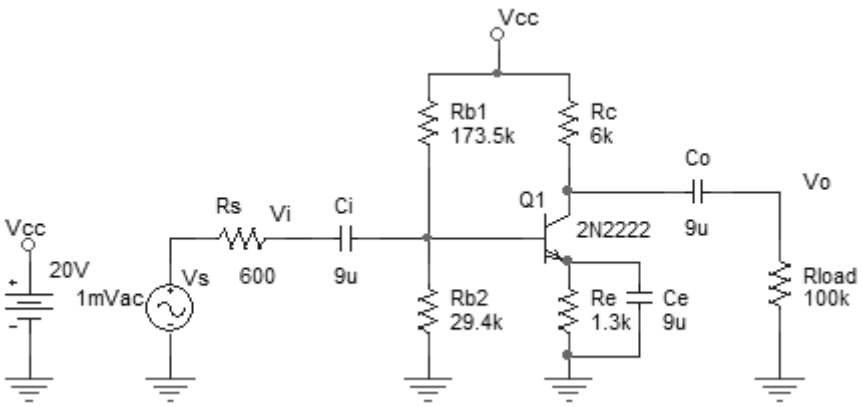
Figura 6.9. – Diagramas de Bode del experimento 4

6.5. Circuitos diseñados manualmente

En esta sección se muestran los resultados de los circuitos diseñados manualmente por un ingeniero electrónico y que serán utilizados para la comparación de los resultados obtenidos del diseño automático con el algoritmo propuesto en este trabajo. Los detalles sobre el proceso de diseño manual, así como la topología seleccionada y el dimensionamiento de los componentes se encuentra en el Apéndice A. Los circuitos obtenidos mediante diseño manual se muestran en la Figura 6.10. En la Tabla 6.10a se encuentran las medidas realizadas sobre el diseño manual obtenido para las condiciones de diseño 1, y en la Tabla 6.10b se encuentran las medidas realizadas sobre el obtenido para las condiciones de diseño 2. Finalmente en la Figura 6.11 se encuentran los diagramas de Bode de la ganancia y de las impedancias de entrada y salida.



(a)



(b)

Figura 6.10. – Circuitos obtenidos mediante diseño manual (a) condiciones de diseño 1 (b) condiciones de diseño 2

(a) Diseño manual 1

$A_v(dB)$ a f_1, f_2 y f_3	$11,34$	$11,34$	$11,34$
$Z_{in}(k\Omega)$ a f_{media}	$25,29$		
$Z_{out}(k\Omega)$ a f_{media}	$5,99$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$9,79$	$-9,87$	
THD	$0,61$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$1,30$	$1,18$	$90,77\,\%$
$Fitness$	$0,58$		

(b) Diseño manual 2

$A_v(dB)$ a f_1 y f_2	$49,15$	$49,17$	
$Z_{in}(k\Omega)$ a f_{media}	$2,23$		
$Z_{out}(k\Omega)$ a f_{media}	$5,52$		
$A_v(f_{low})(dB)$ y $A_v(f_{low}/10)(dB)$	$46,49$	$29,87$	
THD	$3,20$		
$I_C(mA)$, $I_C(127^0)(mA)$, $I_C(127^0)/I_C$	$1,51$	$1,36$	$90,07\,\%$
$Fitness$	$1,523$		

Tabla 6.10. – Medidas de los circuitos diseñados manualmente

6.6. Rendimiento del paralelismo

En esta sección se indica una comparativa del tiempo medio de ejecución utilizando paralelismo y sin utilizarlo, suponiendo la ejecución en un nodo monoprocesador, y la obtención de un factor de mejora por el uso de ejecución paralela.

Considerando que el entorno de ejecución ha sido un cluster de cinco nodos, con un procesador de cuatro núcleos cada uno, se podría suponer que el sistema global consta de 20 unidades de proceso, por lo que el máximo ideal de mejora sería de 20 veces, frente a la ejecución en un nodo monoprocesador. Este límite ideal sólo tiene en cuenta el recurso de CPU, y por lo tanto no tiene en cuenta el uso de otros recursos como el acceso a disco, o el uso de comunicaciones.

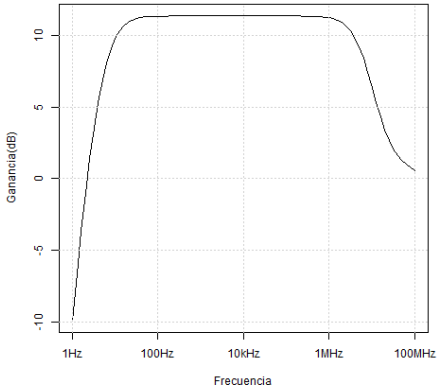
La Tabla 6.11 muestra el tiempo medio de ejecución de un algoritmo evolutivo completo, con los parámetros indicados en la Tabla 6.2, considerando ejecución paralela frente a no paralela. En el caso de ejecución paralela se considera la distribución de procesos indicada en la Tabla 5.1. Finalmente se indica la relación entre ambos

valores de tiempo que representa el factor de mejora por usar paralelismo y que es mayor de 12 veces.

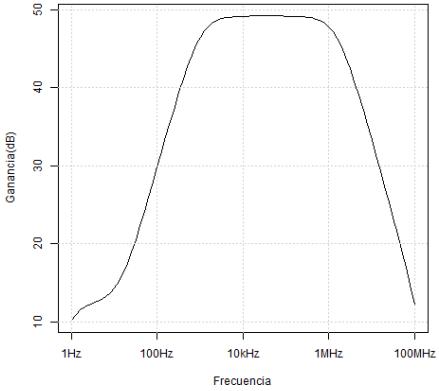
En relación con la distribución de procesos, se ha probado a añadir un proceso más en cada nodo del cluster, sin obtener una mejora en el rendimiento. Por tanto, se ha alcanzado el límite de mejora del programa en su arquitectura actual, siendo necesario replantear una modificación del mismo para mejorar el rendimiento.

Ejecución	Tiempo medio en minutos
No paralela	$95,3'$
Paralela	$7,6'$
Factor mejora	$12,5$

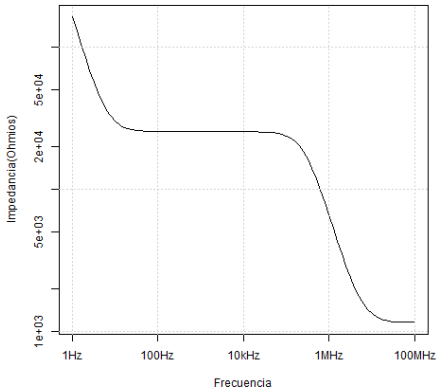
Tabla 6.11. – Rendimiento del paralelismo



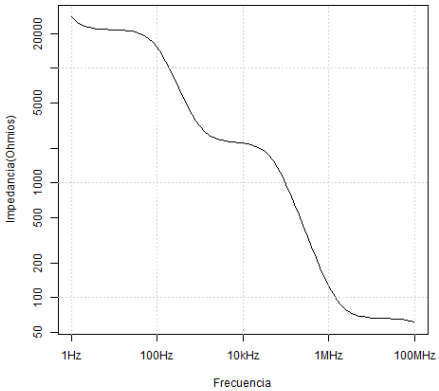
(a) A_v diseño manual 1



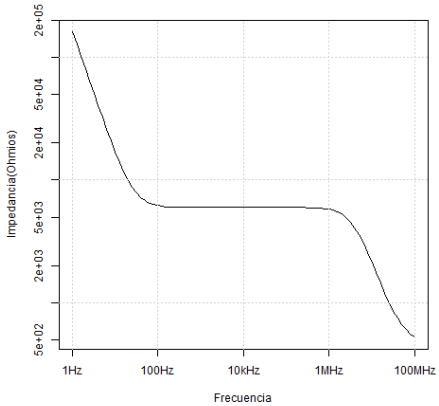
(b) A_v diseño manual 2



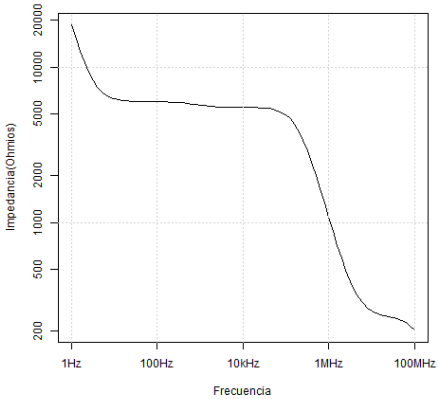
(c) Z_{in} diseño manual 1



(d) Z_{in} diseño manual 2



(e) Z_{out} diseño manual 1



(f) Z_{out} diseño manual 2

Figura 6.11. – Diagramas de Bode de los circuitos diseñados manualmente

7. Discusión y comparación de resultados

En este capítulo se realiza un análisis de los resultados obtenidos, comparando los circuitos diseñados automáticamente con las especificaciones, y posteriormente con los circuitos obtenidos por el diseñador humano. En la primera sección se aborda la discusión sobre las estadísticas de resultados obtenidas a partir de las distintas ejecuciones de cada experimento. Seguidamente se realiza el análisis de los dos mejores circuitos obtenidos en cada experimento. A continuación se comparan los mejores circuitos obtenidos con los circuitos obtenidos por el diseñador humano. Posteriormente se realiza una comparación cualitativa con los resultados obtenidos por Koza y sus colaboradores (Koza et al., 1999a). Finalmente se indican unas consideraciones sobre el efecto engorde.

7.1. Discusión de resultados

En esta sección se aborda la discusión sobre las estadísticas de resultados obtenidas a partir de las distintas ejecuciones de cada experimento. Seguidamente se realiza el análisis de los dos mejores circuitos obtenidos en cada experimento.

Comentarios a la tabla de resultados de la función de adaptación

En la inspección de la Tabla 6.4, se observa que el número de éxitos obtenido en los experimentos 1 y 2 es muy similar, y más bajo que el número de éxitos obtenido en los experimentos 3 y 4. Adicionalmente los valores de MBF han sido mucho más bajos también en los experimentos 3 y 4, por lo que el algoritmo está funcionando mejor para las condiciones de diseño 2, o bien, dichas condiciones de diseño son más fáciles de resolver para el algoritmo planteado.

Atendiendo a los valores mínimos de mejor adaptación obtenidos en cada experimento, los valores más bajos, que corresponden a la obtención de mejores circuitos, corresponden al uso del embrión 2, siendo mejor para las condiciones de diseño 2. En el caso del embrión 1 también se observan mejores resultados para las condiciones de diseño 2.

El experimento 1 es el que peores resultados ha obtenido en términos de mejor adaptación, pues es de un orden de magnitud superior a las mejores adaptaciones obtenidas en los otros tres experimentos.

Los tiempos medios de ejecución del algoritmo varían entre $6,9$ y 8 minutos, no habiendo un patrón claro que se pueda asociar a condiciones de diseño o a embrión.

Comentarios sobre la tabla de longitudes del cromosoma

Se observa que las longitudes medias de las cadenas de bytes son similares en todos los experimentos, excepto en el experimento 4 que muestra una media mayor, las desviaciones típicas también son muy similares a excepción, de nuevo, del experimento 4, seguramente motivado por la limitación del tamaño máximo de cadenas de 250 bytes.

Como se indica posteriormente en la Sección 7.6, se ha observado la aparición del efecto engorde en la longitud de las cadenas de bytes, y por ese motivo se ha puesto la limitación del tamaño máximo anteriormente indicada.

Los valores medios de las longitudes expresadas de los genotipos son todos muy similares, siendo ligeramente mayor la media del experimento 3. Las desviaciones típicas son también muy similares en todos los experimentos, de nuevo, con excepción del experimento 3 que es algo menor.

Para comprobar el efecto del límite máximo de las cadenas en los genotipos, se ha calculado el valor medio de las longitudes expresadas más 3 veces la desviación típica. Suponiendo que la distribución de las longitudes expresadas sea una distribución gaussiana, este valor corresponde a un límite del $99,7\,\%$ de todos los individuos. Observando que este valor es más bajo de 200, lo que corresponde a un $80\,\%$ del límite de las cadenas, no parece que dicho límite esté afectando a la longitud expresada.

En relación con el wrapping, el valor medio de veces que se realiza wrapping varía entre $0,31$ y $0,82$ no existiendo un patrón claro que ligue el valor con las condiciones de diseño o con el tipo de embrión.

Finalmente el número medio de genotipos viables varía entre un $70\,\%$ y un $79\,\%$ de la población final y el de expresables varía entre un $75\,\%$ y un $79\,\%$

Comentarios a las gráficas de evolución de mejor fitness

Las gráficas de evolución del valor de mejor fitness con las generaciones de cada algoritmo, mostradas en la Figura 6.1, muestran la evolución de la mejor ejecución y también la media de las mejores adaptaciones en cada generación del algoritmo.

En general se observa que las curvas medias son descendentes en todos los casos, tendiendo asintóticamente al valor de MBF de cada experimento. Si atendemos a la evolución en el caso de la mejor ejecución se observa que en el experimento 1 se ha

mantenido muy cerca de la media, lo que no ocurre para el resto de experimentos. También se observa que los experimentos realizados sobre las condiciones de diseño 2 convergen antes, lo que junto a los resultados de MBF y SR, da más peso a la conclusión de que dichas condiciones de diseño son más sencillas para el algoritmo.

En los experimentos 2, 3 y 4, se observa un momento en las gráficas en las que la mejor adaptación baja de forma abrupta, lo que no se observa en el caso del experimento 1, en el que un número mayor de generaciones podría haber ayudado a obtener mejores valores de SR y de mejor adaptación.

7.2. Comentarios a los circuitos

En esta sección se van a comentar los circuitos obtenidos, inicialmente en el ámbito de cada experimento y posteriormente en cada condición de diseño.

Experimento 1

Los circuitos 1 y 2 son los mejores obtenidos en el experimento 1. Ambos circuitos presentan una topología muy similar a la seleccionada en el diseño manual. Aparecen algunos grupos de resistencias en serie y en paralelo que son fácilmente simplificables en una sola resistencia. Esto se observa por ejemplo en el colector del circuito 1, y en la base del circuito 2. El circuito 1 presenta una resistencia de emisor y un divisor de tensión en la base. Sin embargo, el circuito 2, a pesar de mostrar una topología similar tiene una resistencia entre la base y masa de $220M\Omega$ lo que hace que el divisor de tensión no esté funcionando como tal. Con ello, la polarización efectiva se produce por las resistencias en serie entre alimentación y la base, así como la resistencia de emisor. Finalmente en el circuito 1 aparece un condensador, del orden de picofaradios, en una de las resistencias del divisor de tensión. Al ser de una capacidad tan baja, sólo tiene efecto en altas frecuencias, por lo que su efecto puede despreciarse en frecuencias medias, es decir, puede considerarse un circuito abierto.

Ambos circuitos alcanzan los valores de ganancia especificados en las tres frecuencias medidas, presentando una respuesta en frecuencia plana en la banda de paso, como se puede comprobar en el diagrama de Bode, Figura 6.3.

En relación con la impedancia de entrada, ambos circuitos cumplen el requisito especificado de forma bastante precisa. Por el contrario, ambos circuitos presentan una impedancia de salida muy similar entre ambos y por debajo de las condiciones de diseño (entre un 12 % y un 14 % de la impedancia total).

Ambos circuitos presentan unos diagramas de Bode de impedancias de entrada y salida muy similares, como se ve en la Figura 6.3.

En relación con la frecuencia de corte, ambos circuitos logran ofrecer la ganancia especificada, lo que indica que han ajustado la frecuencia de corte real a la especificada. La caída de ganancia a una frecuencia una década inferior es muy similar en

ambos circuitos y es mucho más baja de lo especificado, representando una caída de $27dB$ por década.

La distorsión ofrecida por ambos circuitos es muy baja, siendo mucho mejor de lo especificado en ambos circuitos. En el caso del circuito 1, la distorsión es 3 veces más baja de lo exigido, y en el circuito 2, es algo menos de 2 veces más baja de lo exigido.

La corriente de colector está dentro de los márgenes especificados, siendo mayor en el caso del primer circuito que en el segundo. Y también presentan buena estabilidad con la temperatura, siendo su variación de alrededor del 5 % en el caso del primer circuito y menor que el 4 % en el caso del segundo.

Experimento 2

Los circuitos 3 y 4 son los mejores obtenidos en el experimento 2. En ambos circuitos aparecen algunos grupos de resistencias en serie y en paralelo que son fácilmente simplificables en una sola resistencia. Esto se observa por ejemplo en la base y emisor del circuito 3, y en el colector y emisor del circuito 4. El circuito 3 no tiene un divisor de tensión en la base, pues tiene un condensador y una resistencia entre la base y tierra, lo que en continua es un circuito abierto. Con ello, la polarización del circuito 3 se consigue con las resistencias entre la alimentación y la base y las resistencias de emisor. El condensador que se encuentra en la base es cercano a $1nF$ y probablemente sea el causante del diagrama de Bode de impedancia de entrada muy variable y en el que se observan varios polos. El circuito 4 presenta una topología poco habitual, pues dispone de un divisor de tensión que se conecta a la base a través de una tercera resistencia.

Ambos circuitos alcanzan los valores de ganancia especificados en las tres frecuencias medidas, presentando una respuesta en frecuencia plana en la banda de paso, como se puede comprobar en el diagrama de Bode, Figura 6.5.

En relación con la impedancia de entrada, el primer circuito cumple el requisito especificado de forma precisa. Sin embargo el circuito 4 sólo consigue una impedancia mitad de la especificada.

Como se ha indicado previamente el circuito 3 presenta un diagrama de Bode de impedancia de entrada que no es plano en la banda de paso, presentando varios polos. El Bode de impedancia de entrada para el circuito 4 es más plano en casi toda la banda de paso.

En relación con la impedancia de salida, ambos circuitos cumplen el requisito de forma precisa, y presentan diagramas de Bode muy similares y planos en la banda de paso, como se ve en la Figura 6.5.

En relación con la frecuencia de corte, ambos circuitos logran ofrecer la ganancia especificada, lo que indica que han ajustado la frecuencia de corte real a la especificada. La caída de ganancia a una frecuencia una década inferior es mayor en el caso

del circuito 3, con una caída de más de $27dB$ por década. En el caso del circuito 4, la caída no es tan pronunciada como en el circuito 3, siendo de $17dB$ por década.

La distorsión ofrecida por el circuito 4 es muy baja, siendo menos de la mitad de lo exigido. En el caso del circuito 3 la distorsión es buena, siendo un 36 % más baja de lo especificado.

La corriente de colector está dentro de los márgenes especificados, siendo muy similar en ambos circuitos. También presentan muy buena estabilidad con la temperatura siendo menor del 5 % en el caso del circuito 3 y menor de un 11 % en el caso del circuito 4.

Experimento 3

Los circuitos 5 y 6 son los mejores obtenidos en el experimento 3. El circuito 5 es el que menos componentes tiene, no apareciendo ningún conjunto de componentes conectado en serie o paralelo. En caso del circuito 6, sí que aparecen dos resistencias en paralelo en la base y dos condensadores en paralelo en el emisor. En el primer conjunto se puede despreciar el efecto de la resistencia de $220M\Omega$, por ser mucho mayor que la otra de $720k\Omega$. En el segundo conjunto, es posible despreciar el condensador de $580pF$ al ser mucho menor que el otro de $73nF$. En ambos casos la polarización se consigue mediante la resistencia entre alimentación y la base, así como con la resistencia de emisor. En el caso del circuito 6, aparece un condensador de emisor, que como es sabido, permite obtener ganancias más altas, a costa de reducir la impedancia de entrada.

El circuito 5 alcanza los valores de ganancia especificados en las dos frecuencias de forma precisa. El circuito 5 se queda algo bajo, siendo alrededor de $1{,}4dB$ más bajo y alrededor de $1dB$ más bajo en cada frecuencia especificada. Ambos circuitos presentan una banda de paso plana, como se puede observar en el diagrama de Bode, en la Figura 6.7, si bien, el circuito 6 presenta un pico de ganancia a algo por debajo de $1MHz$.

Ambos circuitos alcanzan los valores especificados de impedancias, tanto de entrada como de salida, de una forma precisa. En el diagrama de Bode de impedancias de entrada, el circuito 6 presenta una impedancia de entrada bastante plana, a diferencia del circuito 5 que es descendente con una pequeña meseta alrededor de $10kHz$.

En relación con la impedancia de salida, ambos circuitos cumplen el requisito de forma precisa, y presentan diagramas de Bode muy similares y planos en la banda de paso, como se ve en la Figura 6.7.

En relación con la frecuencia de corte, el circuito 5 alcanza el valor de ganancia indicado, por lo que es la frecuencia de corte exigida. En el caso del circuito 6, la caída es algo menor, siendo algo menos de $1dB$. Con ello, la frecuencia de corte es algo más baja de lo especificado. La caída de ganancia a una frecuencia una década

inferior es mayor en el caso del circuito 5, con una caída de $28dB$ por década. En el caso del circuito 64, la caída no es tan pronunciada como en el circuito 3, siendo de $12,5dB$ por década.

La distorsión ofrecida por el circuito 5 es ajustada, estando muy cerca del umbral. En el caso del circuito 6 la distorsión es buena, siendo un 39 % más baja de lo especificado.

La corriente de colector está dentro de los márgenes especificados, siendo muy similar en ambos circuitos. Ambos circuitos también presentan una buena estabilidad con la temperatura siendo menor, en ambos casos, del 5 % .

Experimento 4

Los circuitos 7 y 8 son los mejores obtenidos en el experimento 4. En ambos circuitos aparecen conjuntos de resistencias en serie y paralelo que podrían simplificarse. En el circuito 7 aparecen en el colector y en la base, y en el circuito 8 en el emisor. Ninguno de los dos circuitos presenta un divisor de tensión en la base, pero sí presentan resistencia de emisor, por lo que la polarización se realiza con la o las resistencias entre alimentación y la base y la resistencia de emisor. Ninguno de los circuitos muestra condensador de emisor, por lo que la ganancia la consiguen con una resistencia de emisor mucho más baja que la resistencia de colector. El circuito 7 presenta una red compleja en el emisor al que incluso se conecta un divisor de tensión. Sin embargo, el valor elevado del paralelo de las resistencias R10 y R11 hace que prácticamente toda la tensión de alimentación caiga en dichas resistencias y, por tanto, se podría simplificar el circuito, eliminando dicho divisor de tensión y conectando a tierra el nodo formado por las resistencias R5, R6 y R7.

Ambos circuitos alcanzan los valores de de ganancia especificados en las dos frecuencias de forma precisa y también ambos circuitos presentan una banda de paso plana, como se puede observar en el diagrama de Bode, en la Figura 6.9.

Los dos circuitos alcanzan los valores especificados de impedancias, tanto de entrada como de salida, de una forma precisa. Ambos circuitos presentan diagramas de Bode de impedancias de entrada muy similares y también de salida, como se ve en la Figura 6.9.

En relación con la frecuencia de corte, ambos circuitos alcanzan el valor de ganancia indicado, por lo que consiguen la frecuencia de corte exigida. La caída de ganancia a una frecuencia una década inferior está dentro de lo exigido siendo mayor de $27dB$ por década en ambos circuitos.

Las distorsiones son iguales, siendo bastante cercanas al valor umbral de lo exigido.

La corriente de colector está dentro de los márgenes especificados, siendo de un orden similar en ambos circuitos. Además, en ambos casos, presenta una buena estabilidad con la temperatura, siendo la variación menor del 5 % en ambos circuitos.

7.3. Comparativa con el diseño manual

Para la comparativa con el diseño manual se consideran los mejores circuitos obtenidos mediante el algoritmo para cada condición de diseño y embrión, que corresponderían a los circuitos 1, 3, 5 y 7.

Condiciones de diseño 1

El circuito que presenta una topología más parecida al diseño manual es el circuito 1, que presenta un divisor de tensión en la base. El circuito 3 tiene una polarización con realimentación de emisor.

Los circuitos 1 y 3 alcanzan la ganancia especificada para las tres frecuencias, no así el diseño manual que se queda casi $0,7dB$ por debajo. Los diagramas de Bode presentan una respuesta en frecuencia plana en la banda de paso en los tres casos.

En relación con la impedancia de entrada, los circuitos 1 y 3 alcanzan el valor especificado de una forma muy precisa, no así en el caso del diseño manual, que supera la impedancia de entrada especificada en más de un 25 %.

El diagrama de Bode de la impedancia de entrada del circuito 1 es muy similar al del diseño manual. En el caso del circuito 3, la impedancia no es plana.

En relación con la impedancia de salida, tanto los circuitos 1 y 3, como el diseño manual alcanzan la impedancia de salida de forma muy precisa. El Bode de la impedancia de salida es similar en los tres casos.

Los circuitos 1 y 3 alcanzan la ganancia especificada en la frecuencia de corte, siendo algo más alta en el caso del diseño manual, por lo que la frecuencia de corte real en este caso es algo menor de la especificada. Los tres circuitos cumplen la condición de caída una década más baja. En el caso de los circuitos 1 y 3, la caída es mayor de $27dB$ por década mientras que en el caso manual la caída es algo menor de $20dB$ por década

La distorsión ofrecida por el circuito 1 es muy buena siendo una tercera parte de lo exigido. La distorsión ofrecida por el circuito 3 es muy parecida a la distorsión del diseño manual, siendo de un tercio menos de lo especificado.

La corriente de colector en los tres circuitos está dentro del margen especificado, siendo mayor en el caso del circuito 1. En cuanto a la estabilidad con la temperatura, el circuito 3 es el más estable con una variación algo superior a un 4 %, el circuito 1 varía alrededor de un 5 %, y el diseño manual varía casi un 10 %.

Finalmente, en relación con el valor de adaptación, el mejor circuito de estos tres es el circuito 3, seguido por el diseño manual y finalmente el circuito 1.

Condiciones de diseño 2

Los circuitos 5 y 7 tienen una topología distinta al diseño manual. En primer lugar no disponen de divisor de tensión en la base, realizando la polarización con realimentación de emisor. Otra diferencia es la falta del condensador de emisor.

Los circuitos 5 y 7 alcanzan la ganancia especificada, de una forma muy precisa el segundo y ligeramente por encima en el caso del primero. El diseño manual da una ganancia ligeramente más de un 1dB mayor. Los diagramas de Bode presentan una respuesta en frecuencia plana en la banda de paso en los tres casos.

En relación con la impedancia de entrada, los circuitos 5 y 7 alcanzan el valor especificado de una forma muy ajustada, mientras que el diseño manual supera la impedancia de entrada en un 10 %.

El diagrama de Bode de la impedancia de entrada de los circuitos 5 y 7 es muy similar. En el caso del diseño manual, éste no es tan plano, presentando dos polos.

En relación con la impedancia de salida, los circuitos 5 y 7 alcanzan la impedancia de salida de forma muy precisa. El diseño manual se queda ligeramente por debajo. El Bode de la impedancia de salida del diseño manual es más plano que en el caso de los circuitos 5 y 7.

Los circuitos 5 y 7 alcanzan la ganancia especificada en la frecuencia de corte, siendo algo más alta en el caso del diseño manual, por lo que la frecuencia de corte real en este caso es algo menor de la especificada. Los tres circuitos cumplen la condición de caída una década más baja. En el caso de los circuitos 5 y 7, la caída es mayor de $27dB$ por década mientras que en el caso manual la caída es mayor de $16dB$ por década

La distorsión ofrecida por los circuitos 5 y 7 está dentro de lo exigido siendo algo mejor la del circuito 7. La distorsión del diseño manual no cumple el requisito exigido, ofreciendo un valor de $3,20\,\%$.

La corriente de colector en los tres circuitos está dentro del margen especificado, siendo de un orden similar en los tres casos. En cuanto a la estabilidad con la temperatura, los circuitos 5 y 7 varían menos de un 5 %, mientras que el diseño manual varía menos de un 10 %.

Finalmente, en relación con el valor de adaptación, el mejor circuito de estos tres es el circuito 7, seguido por el circuito 5 y finalmente el diseño manual.

7.4. Comparación con los resultados obtenidos por Koza

Dado que los trabajos más parecidos al aquí propuesto corresponden a los realizados por Koza y sus colaboradores (Koza et al., 1999a), se realiza aquí una comparativa

entre ambas propuestas. Aunque dichos trabajos también están relacionados con el diseño automático de amplificadores, lamentablemente, no abordan los mismos objetivos que los aquí planteados, por lo que la comparación sólo se realiza de forma cualitativa.

Concretamente en los trabajos de Koza y sus colaboradores se aborda el diseño automático de amplificadores para diferentes ganancias utilizando funciones de adaptación distintas en cada uno de los casos. A continuación se describen algunas de las funciones de adaptación utilizadas, observando que la ganancia aparece en todas ellas, diferenciándose en la inclusión de otros parámetros o en la forma de realizar las medidas:

1. Función de adaptación con objetivo de ganancia, siendo la adaptación la suma de las desviaciones de la ganancia especificada, realizando medidas en 151 muestras dentro del rango de frecuencias $20Hz$ a $20{,}000Hz$.
2. Función de adaptación con objetivos de rechazo a la continua, ganancia y baja distorsión, cuyas medidas se obtienen del análisis de Fourier con una señal sinusoidal de entrada a $1kHz$.
3. Función de adaptación con objetivos de ganancia, bias (voltaje de salida 0) y linealidad mediante un análisis en el dominio del tiempo.

Koza consigue buenos resultados de diseño de amplificadores que le permiten afirmar que los diseños obtenidos son competitivos frente a los diseños realizados por humanos.

La función de adaptación presentada en este trabajo se basa en un mayor número de especificaciones de diseño del amplificador, por otra parte más habituales en el diseño tradicional de un amplificador, como son ganancia, impedancias de entrada y salida, frecuencias de corte y caída por década, distorsión, polarización (corriente de colector) y estabilidad con la temperatura. Por tanto ni las especificaciones de diseño, ni las funciones de adaptación son directamente comparables.

Podría decirse que aunque el objetivo de Koza es diseñar amplificadores más complejos que los aquí abordados, desde el punto de vista del número de etapas de transistores implicadas en el diseño, el número de especificaciones impuestas en sus diseños es menor que el usado en los diseños aquí abordados. Es decir, por un lado, desde el punto de vista de la topología, Koza plantea diseños de amplificadores más complejos, pero, por otro lado, desde el punto de vista de las especificaciones de diseño, dichos diseños son menos exigentes que los aquí considerados. También es preciso señalar que, con los recursos computacionales disponibles de los que aquí se ha dispuesto, hubiese sido muy difícil abordar los diseños planteados por Koza, donde se llegan a realizar ejecuciones con poblaciones de dos millones de individuos, frente a las poblaciones aquí utilizadas de 1000 individuos.

Otra de las diferencias es que Koza permite que el algoritmo evolutivo pueda añadir transistores, por lo que contempla una función de creación de transistores dentro de las funciones de creación de componentes. En la propuesta realizada en este trabajo

se quería que conocimiento del dominio jugara un papel mayor, por lo que se ha impuesto el uso de un sólo transistor y su configuración en emisor común, utilizando un embrión con dichas características.

Debido a la posibilidad de introducir transistores, los resultados de Koza presentan siempre más de un transistor en cada circuito. Además el mejor circuito para un amplificador de $10dB$ consta de dos etapas, no siendo directamente comparable con las topologías ni con las especificaciones de diseño que se han considerado en este trabajo.

7.5. Rol del conocimiento del dominio

El algoritmo evolutivo se ha mostrado eficaz en el diseño automático de amplificadores electrónicos, a pesar de no disponer de conocimiento experto en el dominio de la electrónica. Tal como indica Koza, el algoritmo evolutivo no tiene en concreto conocimiento sobre las leyes de Kirchoff, las ecuaciones del transistor, modelo equivalente a frecuencias medias del mismo, análisis de Fourier, topologías de circuito ni dimensionamiento de componentes. Conocimiento necesario para que un diseñador humano pueda realizar un proceso de diseño manual. De hecho, el único conocimiento del dominio que se inyecta en la aproximación propuesta es utilizado en el diseño de los dos embriones descritos en la Sección 4.7. Dicho conocimiento está relacionado con el hecho de que el amplificador a diseñar se podría obtener con una única etapa de transistor bipolar en emisor común. Conocimiento, que por otro lado, también utiliza el humano en sus dos diseños manuales.

Se podría decir que las funciones de transformación descritas en la Sección 4.1 son una forma de inyectar conocimiento del dominio. Sin embargo, el conocimiento inyectado en estas reglas de transformación es genérico, es decir, proviene de la electrónica. Por tanto, podrían usarse no sólo en del diseño de los amplificadores electrónicos, sino en el diseño de cualquier otro tipo de circuito analógico.

Finalmente, hay determinado conocimiento del dominio que se usa pero, en contra de lo que pudiera parecer, no se inyecta de forma directa en el algoritmo evolutivo y es el siguiente:

1. Estructura fija, que dispone de entrada y salida, fuente de alimentación, fuente de señal, resistencias de fuente de señal y de carga, y se muestra en la 4.9a. Esto forma parte de las condiciones de contorno del problema pero no afecta de forma directa a la dinámica intrínseca del proceso evolutivo. Es decir, si las condiciones de contorno fueran otras, los mecanismos implicados en el proceso evolutivo seguirían siendo los mismos.
2. Función de adaptación, que incluye parámetros de ganancia, impedancias de entrada y salida, frecuencia de corte y caída, distorsión, corriente de colector y estabilidad con la temperatura. Esto no interviene de forma directa en el proceso evolutivo, interviene sólo en cómo medir la bondad de un individuo. Está

claro que la propia medida determina el proceso de selección de los más aptos, pero, independientemente del valor de la medida, los mecanismos implicados en el proceso evolutivo siguen siendo los mismos.

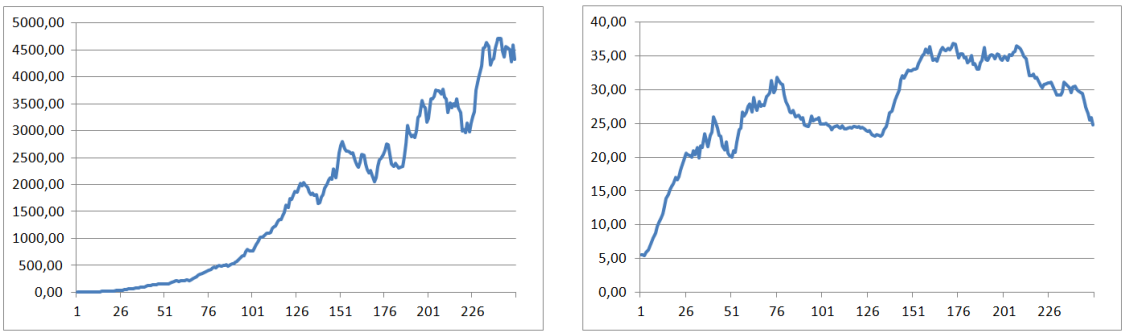
3. Tipos de análisis a realizar con Ngspice para obtener las medidas, que incluyen análisis de pequeña señal, análisis transitorio y de Fourier, y análisis a diferentes temperaturas. Al igual que en el caso, anterior, esto sólo influye en cómo se hace la medida.

7.6. Efecto engorde

El efecto engorde presentado en la Sección 3.5 al hablar de Programación Genética, aparece también en esta implementación, dando lugar al crecimiento de la longitud de las cadenas con el número de generaciones. En esta sección se describe un mecanismo simple para reducir este efecto.

En primer lugar definimos longitud expresada de una cadena de bytes, como el número de bytes o codones empleados hasta obtener una expresión completa. Contraponemos dicha longitud contra la longitud total de la cadena de bytes.

Como ejemplo se muestra la evolución de estos valores en una ejecución del algoritmo para desarrollo de circuitos. La Figura 7.1a muestra la evolución del valor medio de las longitudes totales de las cadenas de bytes durante la ejecución del algoritmo y como se puede observar su tendencia es creciente. En la Figura 7.1b se muestra la evolución de la media de las longitudes expresadas y como se puede observar inicialmente crece, pero se mantiene más estable. En este caso, la relación entre ambos valores es algo mayor de 100.



(a) **(b)**

Figura 7.1. – Efecto engorde: (a) longitud total media (b) longitud expresada media

Este efecto tiene un impacto en el rendimiento del algoritmo, pues se está realizando el tratamiento de cadenas de bytes de un tamaño mayor de la longitud expresable. Este impacto afecta al tiempo de ejecución y a la memoria utilizada. En nuestro

caso la memoria no causa problemas, pero sí existe una limitación importante en tiempo de ejecución, por lo que resulta interesante añadir un mecanismo que evite el efecto engorde.

Por ello, en esta implementación se ha añadido una limitación del tamaño máximo de las cadenas a 250 bytes. Esta limitación ha sido utilizada en los cuatro experimentos mostrados.

8. Conclusiones y futuros trabajos

En este trabajo se ha presentado un algoritmo para la síntesis automática de amplificadores analógicos de una etapa basados en *Grammatical Evolution* (GE) y en las expresiones de desarrollo de Koza. GE es un algoritmo evolutivo capaz de generar código en cualquier lenguaje de programación, que utiliza cadenas de bytes de longitud variable. El proceso de decodificación de un cromosoma hace uso de una gramática BNF del lenguaje objetivo y se basa en recorrer la cadena de bytes, determinando cada uno de ellos qué regla de producción de la gramática será utilizada. Este proceso da lugar a una traducción de genotipo a fenotipo, generando una expresión perteneciente a la gramática utilizada.

En los trabajos de Koza y sus colaboradores presentaron una forma de representación de circuitos mediante árboles de parseado. Este tipo de representación, llamada de desarrollo, se basa en aplicar una sucesión de funciones de transformación sobre un circuito de partida, llamado embrión, y como resultado de este proceso de desarrollo, se obtiene un circuito final. El algoritmo desarrollado en este trabajo, basado en GE, utiliza una gramática compuesta basada en el conjunto de las funciones de transformación definidas por Koza. Hasta donde se sabe por la bibliografía relacionada con el área, la gramática aquí propuesta es la primera dedicada a abordar el diseño automático de circuitos analógicos mediante el uso de GE.

Se ha trabajado también en la creación de una función de adaptación compuesta por diferentes términos que permite hacer medible computacionalmente las especificaciones de diseño del amplificador objetivo.

Se ha desarrollado un programa en *Java* a medida para implementar el algoritmo propuesto. Se ha utilizado también la herramienta *JavaCC* para la implementación del parser de las expresiones de desarrollo. El programa se ha paralelizado para su ejecución en un cluster de cinco nodos, lo que ha permitido reducir el tiempo de ejecución. La paralelización ha atendido a la evaluación de los circuitos que es la parte más costosa computacionalmente del algoritmo. La simulación de los circuitos se ha realizado con la herramienta *Ngspice*.

Se han analizado dos casos de estudio, correspondientes al diseño de dos amplificadores diferentes. Para cada diseño, se han utilizados dos embriones, dando lugar a cuatro experimentos diferentes. Se han mostrado los resultados generales de cada experimento, mostrando medidas estándar de algoritmos evolutivos, como SR y MBF, así como tiempo medio de ejecución. También se han mostrado estadísticas de longitud del cromosoma y las curvas de evolución de la mejor adaptación y fi-

nalmente se han mostrado los dos mejores circuitos obtenidos en cada experimento junto con sus medidas y sus diagramas de Bode.

En la discusión de resultados se ha visto que a pesar de que las condiciones de diseño del segundo amplificador dan lugar a una topología algo más compleja en un diseño tradicional, se ha obtenido un número mayor de éxitos así como un mejor MBF, por lo que dichas condiciones resultan más fáciles de resolver para el algoritmo que las condiciones de diseño del primer amplificador.

El tamaño del cromosoma se ha limitado a 250 bytes para evitar el efecto engorde que se había observado previamente. Esta medida ha resultado eficaz, permitiendo obtener buenos resultados del algoritmo. También se ha estimado que la longitud expresada de la mayoría de los cromosomas es bastante inferior a dicho límite.

La topología de los circuitos obtenidos por el algoritmo evolutivo suelen mostrar polarización con realimentación de emisor, apareciendo un divisor de tensión en la base sólo en algunos casos. También se ha observado que, en dichos circuitos, aparecen conjuntos de resistencias y/o condensadores, en serie o paralelo, que podrían simplificarse posteriormente, dando lugar, respectivamente, a una resistencia o capacidad equivalente. En otros casos, se han obtenido ramas que poseen condensadores de capacidades muy pequeñas. Dichas ramas se pueden eliminar en el circuito por considerarse que, a frecuencias medias, dichos condensadores representan circuitos abiertos.

El algoritmo evolutivo ha logrado conseguir, en casi todos los casos, las especificaciones de ganancia de una forma muy precisa. Las impedancias de entrada y salida también las ha conseguido de una forma muy precisa para el diseño del segundo amplificador. En el caso de diseño del primer amplificador con el embrión 1, no se han conseguido las impedancias de salida, y con el embrión 2, no se ha conseguido la impedancia de entrada en el segundo mejor circuito.

El algoritmo evolutivo también ha conseguido la frecuencia de corte y la caída de forma muy precisa, con una excepción en el diseño del segundo amplificador con el embrión 1.

La especificación de distorsión se ha conseguido en todos los diseños evolutivos, obteniendo valores, como en el caso del diseño del primer amplificador, muy por debajo del umbral especificado.

Todos los circuitos evolucionados han conseguido una corriente de colector dentro del margen exigido y han mostrado una buena estabilidad, con muchos casos alrededor de sólo un 5 % de variación.

El objetivo de Koza es el diseño de amplificadores más complejos que los aquí abordados, requiriendo mayores recursos computacionales para permitir el uso de poblaciones de dos millones de individuos, frente a las poblaciones de 1000 individuos, aquí utilizadas. Sin embargo, las funciones de adaptación utilizadas por Koza exigen un menor número de especificaciones que las aquí propuestas, que son, por otra parte, más habituales en el diseño tradicional de un amplificador, como son la impedancia

de entrada y salida, frecuencias de corte y caída por década, polarización (corriente de colector) y estabilidad con la temperatura. Siendo los objetivos diferentes, no ha sido posible realizar una comparación cuantitativa con los resultados de Koza. Por tanto, la comparación se ha realizado frente al diseño realizado por un diseñador humano.

En la comparativa con un diseñador humano, se observa que el algoritmo evolutivo en general ha conseguido las especificaciones de forma más precisa. El hecho de que el diseño manual sea aproximado se debe a a que el transistor es un elemento no lineal y para poder resolver el diseño se recurre a aproximaciones, siendo necesario realizar varias iteraciones hasta ajustar el circuito a las especificaciones en la medida de lo posible.

El algoritmo evolutivo se ha mostrado eficaz en el diseño automático de amplificadores electrónicos, a pesar de no disponer de conocimiento experto en el dominio de la electrónica, como el que usa un diseñador humano en la realización de su diseño. El único conocimiento del dominio que se inyecta está en los embriones utilizados. Las funciones de transformación conllevan un conocimiento genérico, pero no es específico de diseño de amplificadores. Finalmente, hay determinado conocimiento del dominio que se usa pero no de forma directa en el algoritmo evolutivo, en relación con la estructura fija, la función de adaptación y los tipos de análisis a realizar con el simulador de circuitos.

Como trabajo futuro se comentan diferentes propuestas ordenadas de menor a mayor complejidad. Así, en primer lugar, se podría empezar por permitir que sea el propio proceso evolutivo el que elija la configuración del transistor, emisor, base o colector común, en función de las especificaciones de diseño. También se podría estudiar el uso de un embrión más simple, y cercano a los utilizados por Koza (Koza et al., 1999a), lo que permitiría comprobar la capacidad del algoritmo evolutivo cuando el espacio de búsqueda es más amplio. Otras posibilidades podrían contemplar el diseño de amplificadores con más de una etapa, y el uso de realimentación.

A. Diseño manual de un amplificador

En este apéndice se describe cómo se realiza el proceso de diseño manual de un amplificador que cumpla las condiciones de diseño 1, del que se describirán las asunciones y aproximaciones realizadas, así como las ecuaciones que lo rigen. Posteriormente se indican las consideraciones y modificaciones para el caso diseño de un amplificador que cumpla las condiciones de diseño 2.

A.1. Diseño manual del amplificador 1

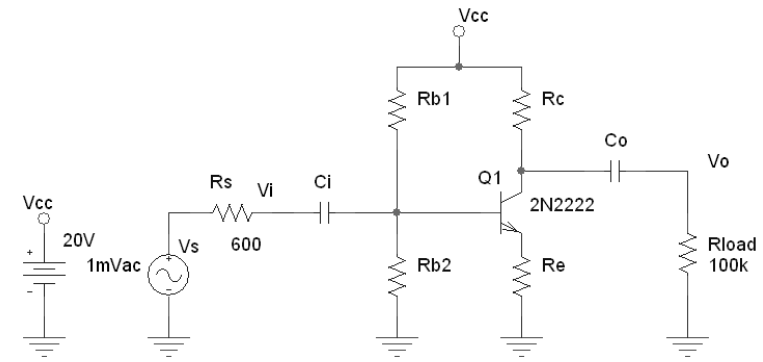
En primer lugar se indicarán las asunciones del diseñador, que permiten seleccionar una topología adecuada, que, para este diseño, se muestra en la Figura A.1a. Las asunciones son las siguientes:

1. Dadas las especificaciones de diseño del amplificador, que se recogen en la Tabla 6.1a, se sabe que una etapa en emisor común con un transistor BJT podría cumplir dichas especificaciones, pues tiene las siguientes características:
   a) Ganancia del orden de unidades de dB
   b) Impedancia de entrada del orden de decenas de kΩ
   c) Impedancia de salida del orden de unidades de kΩ
2. En la polarización del transistor se elige la polarización mediante divisor de tensión porque es la que mejor estabiliza los posibles cambios de ganancia de corriente, que es muy sensible a la temperatura.
3. El diseño se hará suponiendo condiciones de trabajo en frecuencias bajas e intermedias, lo que implica una simplificación del modelo equivalente del transistor en pequeña señal en este rango de frecuencias.

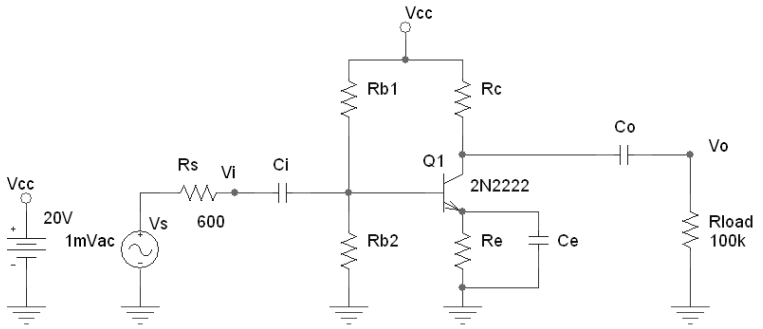
Las ecuaciones y simplificaciones de este análisis se encuentran con mayor detalle en (Rincón and Carmona, 2004).

Análisis en gran señal (polarización) del amplificador

Este análisis se realiza en continua y permite obtener el punto de polarización del transistor *Q,* que vendrá dado por la corriente de colector I_{CQ} y la tensión entre emisor y colector V_{CEQ}.



(a)



(b)

Figura A.1. – Topologías para el diseño manual (a) condiciones de diseño 1 (b) condiciones de diseño 2

En este análisis que se realiza en continua, se asume que los condensadores se comportan como circuitos abiertos, por lo que el circuito queda simplificado, teniendo efecto únicamente el transistor y las resistencias R_{b1}, R_{b2}, R_e y R_c. Las ecuaciones que se derivan de este análisis son:

$$V_{cc} = I_{CQ}R_c + V_{CEQ} + I_E R_e \tag{A.1}$$

$$V_{TH} = I_B R_{TH} + V_{BE} + I_E R_e \tag{A.2}$$

$$donde\, V_{TH} = \frac{V_{cc}R_{b2}}{R_{b1} + R_{b2}}\, y\, R_{TH} = R_{b1} \parallel R_{b2}$$

Donde V_{cc} es la tensión de alimentación, V_{TH} y R_{TH} son la tensión y la resistencia del equivalente de Thévenin del divisor de tensión. V_{BE} es la tensión entre la base y emisor, e I_B e I_E son la corriente de base y de emisor respectivamente.

Para que el transistor amplifique es necesario que trabaje siempre en zona activa, por lo que se cumplirán las siguientes ecuaciones:

$$I_{CQ} = \beta\, I_B \tag{A.3}$$

$$V_{BE} \simeq 0{,}7V \tag{A.4}$$

Donde β es la ganancia de corriente en gran señal.

Análisis en pequeña señal

En este análisis se asume que el amplificador trabaja a frecuencias medias en las que los condensadores se comportan como cortocircuitos. También se utiliza el modelo equivalente en pequeña señal del transistor, simplificado también para frecuencias medias. Este modelo se muestra en la Figura A.2a y presenta las siguientes ecuaciones:

$$r_\pi = \frac{h_{FE} V_T}{I_{CQ}} \tag{A.5}$$

$$h_{FE} \simeq \beta \tag{A.6}$$

$$con\ V_T = 25mV$$

Donde $r_\pi$es la resistencia de entrada en la base en pequeña señal, h_{FE} es la ganancia de corriente en pequeña señal, que es aproximadamente igual a la ganancia de corriente en gran señal β y V_T es el voltaje térmico calculado a temperatura ambiente.

Finalmente el circuito equivalente en pequeña señal del circuito completo se muestra en la Figura A.2b del que se derivan las siguientes ecuaciones:

$$A_v = -\frac{h_{FE}(R_C \parallel R_E)}{r_\pi + (1 + h_{FE})R_e} \tag{A.7}$$

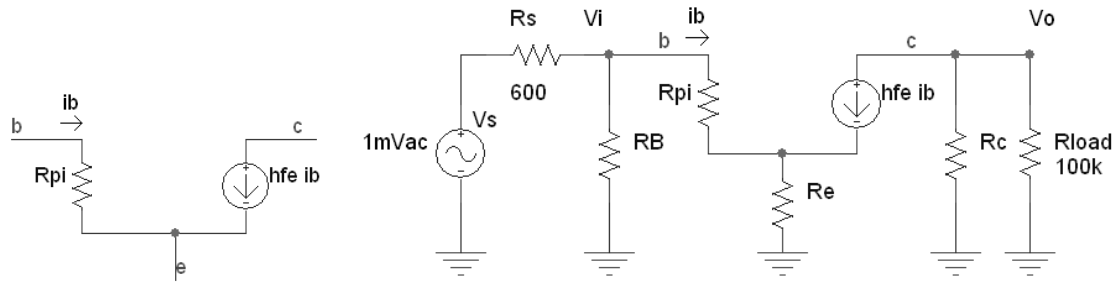
$$Z_{in} = R_B \parallel [r_\pi + (1 + h_{FE})\, R_e] \tag{A.8}$$

$$Z_{out} = R_c \tag{A.9}$$

Finalmente se utiliza el método de las constantes de tiempo para obtener la frecuencia inferior de corte. Adicionalmente y para mayor simplicidad, se asume que los condensadores son de igual valor, lo que da lugar a la ecuación indicada en Ecuación A.11.

$$C_i = C_o \tag{A.10}$$

$$f_{low} = \frac{1}{2\pi C_i}\left(\frac{1}{Z_{in} + R_s} + \frac{1}{Z_{out} + R_{load}}\right) \tag{A.11}$$



(a) Modelo equivalente del transistor

(b) Modelo equivalente del circuito

Figura A.2. – Análisis en pequeña señal

Restricciones de diseño

Para conseguir la máxima excursión de la señal de salida sin distorsión es necesario que se cumpla la condición indicada en Ecuación A.12.

$$Si\ R_{load} \gg R_C \rightarrow V_{CEQ} \simeq \frac{V_{cc}}{2} \tag{A.12}$$

Hasta aquí se dispone de 11 ecuaciones con 12 incógnitas, por tanto se puede añadir una suposición más para disponer de una ecuación adicional. Para estabilizar dicha tensión, se exige como criterio de diseño, que la corriente de base que se deriva del divisor sea menos de un 10 % de la corriente que atraviesa el divisor.

Finalmente mediante un proceso que puede implicar varias iteraciones, se obtiene los valores finales de cada uno de los componentes del circuito de la Figura A.1a y que vienen resumidos en la Tabla A.1a.

A.2. Diseño manual del amplificador 2

Para este diseño se utiliza una topología diferente que se muestra en la Figura A.1b. Esta topología incluye un condensador de emisor que cortocircuita la resistencia de emisor en frecuencias medias, lo que permite obtener ganancias más grandes, con la contrapartida de disminuir la impedancia de entrada. El análisis en gran señal es idéntico al realizado en el amplificador para las condiciones de diseño 1, y en el caso de pequeña señal se considera que la R_e queda cortocircuitada en frecuencias medias por C_e, quedando como aparece en la Figura A.3 y del que se derivan las siguientes ecuaciones para la ganancia, la impedancia de entrada y la de salida:

$$A_v = -\frac{h_{FE}(R_C \parallel R_{load})}{r_\pi} \tag{A.13}$$

$$Z_{in} = R_B \parallel r_\pi \tag{A.14}$$

$$Z_{out} = R_c \tag{A.15}$$

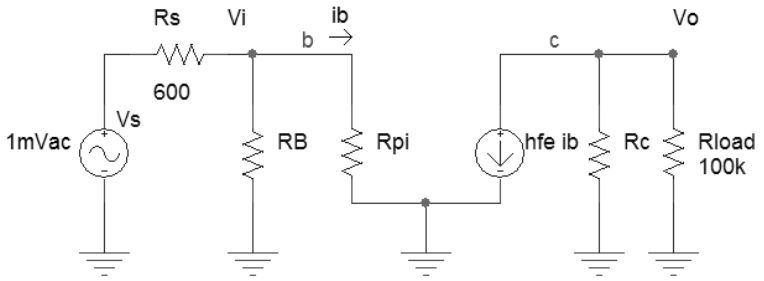


Figura A.3. – Modelo equivalente del circuito para las condiciones de diseño 2

Otros aspectos donde influye dicho condensador es en la frecuencia inferior de corte, como se ve en la ecuación Ecuación A.17, y sobre todo en el margen dinámico, donde la condición para conseguir el máximo margen dinámico, la ecuación Ecuación A.12 se ve modificada quedando como se indica en la ecuación Ecuación A.18.

$$C_i = C_o = C_e \tag{A.16}$$

$$f_{low} = \frac{1}{2\pi C_i}\left(\frac{1}{Z_{in}+R_s} + \frac{1}{Z_{out}+R_{load}} + \frac{1}{R_e \parallel \left[\frac{r_\pi+(R_B\|R_S)}{1+h_{FE}}\right]}\right) \tag{A.17}$$

$$Si\ R_{load} \gg R_C \rightarrow V_{CEQ} \simeq \frac{V_{cc} - I_c R_e}{2} \tag{A.18}$$

Finalmente mediante un proceso que puede implicar varias iteraciones, se obtiene los valores finales de cada uno de los componentes del circuito de la Figura A.1b y que vienen resumidos en la Tabla A.1b.

(a) Condiciones de diseño 1

Componente	Valor
R_c	$6k\Omega$
R_e	$1{,}5k\Omega$
R_{b1}	$195k\Omega$
R_{b2}	$33,8k\Omega$
C_i	$1\mu F$
C_o	$1\mu F$

(b) Condiciones de diseño 2

Componente	Valor
R_c	$6k\Omega$
R_e	$1,3k\Omega$
R_{b1}	$173,5k\Omega$
R_{b2}	$29,4k\Omega$
C_i	$9\mu F$
C_o	$9\mu F$

Tabla A.1. – Componentes resultado del diseño manual

Bibliografía

Aaserud, O. and Nielsen, I. R. (1995). Trends in current analog design-a panel debate. *Analog Integrated Circuits and Signal Processing*, 7(1):5–9.

Ando, S. and Iba, H. (2000). Analog circuit design with a variable length chromosome. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 994 –1001 vol.2.

Antao, B. (1996). Trends in cad of analog ics. *Circuits and Devices Magazine, IEEE*, 12(5):31–41.

Camenzind, H. (2005). *Designing analog chips*. Virtual Bookworm. Com Pub Incorporated.

Coello Coello, C. A. and Aguirre, A. H. (2002). Design of combinational logic circuits through an evolutionary multiobjective optimization approach. *Artif. Intell. Eng. Des. Anal. Manuf.*, 16(1):39–53.

Eiben, A. E. and Smith, J. E. (2008). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer.

Freeman, E., Freeman, E., Bates, B., and Sierra, K. (2004). *Head First Design Patterns*. O' Reilly & Associates, Inc.

Gan, Z., Yang, Z., Shang, T., Yu, T., and Jiang, M. (2010). Automated synthesis of passive analog filters using graph representation. *Expert Systems with Applications*, 37(3):1887 – 1898.

Gielen, G. and Rutenbar, R. (2000). Computer-aided design of analog and mixed-signal integrated circuits. *Proceedings of the IEEE*, 88(12):1825 –1854.

Grimbleby, J. (2000). Automatic analogue circuit synthesis using genetic algorithms. *Circuits, Devices and Systems, IEE Proceedings* -, 147(6):319 –323.

Grimbleby, J. B. (1995). Automatic analogue network synthesis using genetic algorithms. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 53–58. IET.

Harjani, R., Rutenbar, R. A., and Carley, L. R. (1987). A prototype framework for knowledge-based analog circuit synthesis. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, DAC '87, pages 42–49, New York, NY, USA. ACM.

Higuchi, T., Iwata, M., Kajitani, I., Yamada, H., Manderick, B., Hirao, Y., Murakawa, M., Yoshizawa, S., and Furuya, T. (1996). Evolvable hardware with genetic learning. In *Circuits and Systems, 1996. ISCAS '96., Connecting the World., 1996 IEEE International Symposium on*, volume 4, pages 29 –32 vol.4.

ISO/IEC-14977 (1996). Information technology – syntactic metalanguage – extended bnf.

Karpuzcu, U. R. (2005). Automatic verilog code generation through grammatical evolution. In *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 394–397. ACM.

Koza, J., III, F. H. B., Andre, D., and Keane, M. A. (1998). Evolutionary design of analog electrical circuits using genetic programming. In *in I. C. Parmee (ed.),Adaptive Computing in Design and Manufacture*, pages 177–192. Springer Verlag.

Koza, J. R., Andre, D., Bennett, F. H., and Keane, M. A. (1999a). *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition.

Koza, J. R., III, F. H. B., Andre, D., and Keane, M. A. (1999b). The design of analog circuits by means of genetic programming.

Koza, J. R., III, F. H. B., Andre, D., and Keane, M. A. (2000a). Automatic design of analog electrical circuits using genetic programming. In Cartwright, H., editor, *Intelligent Data Analysis in Science*, chapter 8, pages 172–200. Oxford University Press, Oxford.

Koza, J. R., III, F. H. B., Andre, D., and Keane, M. A. (2000b). Synthesis of topology and sizing of analog electrical circuits by means of genetic programming.

Martens, E. and Gielen, G. (2008). Classification of analog synthesis tools based on their architecture selection mechanisms. *Integration, the VLSI Journal*, 41(2):238 – 252.

Mattiussi, C. (2005). *Evolutionary synthesis of analog networks*. PhD thesis, Università degli Studi di Trieste.

Mattiussi, C. and Floreano, D. (2007). Analog genetic encoding for the evolution of circuits and networks. *Evolutionary Computation, IEEE Transactions on*, 11(5):596 –607.

Mühlenbein, H., Zinchenko, L., Kureichik, V., and Mahnig, T. (2007). Effective mutation rate for probabilistic evolutionary design of analogue electrical circuits. *Applied Soft Computing*, 7(3):1012 – 1018.

Nagel, L. W. and Pederson, D. O. (1973). *SPICE: Simulation program with integrated circuit emphasis*. Electronics Research Laboratory, College of Engineering, University of California.

Nenzi, P. and Vogt, H. (2011). Ngspice users manual version 23.

O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., and Brabazon, A. (2008). Geva: grammatical evolution in java. *SIGEVOlution*, 3(2):17–22.

O'Neill, M. and Ryan, C. (1999a). Genetic code degeneracy: Implications for grammatical evolution and beyond. In *Advances in Artificial Life*, pages 149–153. Springer.

O'Neill, M. and Ryan, C. (1999b). Under the hood of grammatical evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1143–1148.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5:349–358.

O'Neill, M. and Ryan, C. (2004). Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. *Genetic Programming*, pages 138–149.

O'Neill, M., Ryan, C., Keijzer, M., and Cattolico, M. (2003). Crossover in grammatical evolution. *Genetic Programming and Evolvable Machines*, 4:67–93. 10.1023/A:1021877127167.

Puhan, J., Tuma, T., and Fajfar, I. (1999). Optimisation methods in spice: a comparison. In *Proceedings of European Conference on Circuit Theory and Design. ECCTD'99. Vol*, volume 2, pages 1279–1282.

Rincón, M. and Carmona, E. (2004). *Prácticas de electrónica analógica lineal*. Universidad Nacional de Educación a Distancia.

Rutenbar, R. (1993). Analog design automation: Where are we? where are we going? In *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, pages 13.1.1 –13.1.7.

Ryan, C., Nicolau, M., and O'Neill, M. (2002). Genetic algorithms using grammatical evolution. In *Genetic Programming*, pages 278–287. Springer.

Ryan, C. and O'Neill, M. (1998). Grammatical evolution: A steady state approach. *Late Breaking Papers, Genetic Programming*, 1998:180–185.

Ryan, C., O'Neill, M., and Collins, J. (1998). Grammatical evolution: solving trigonometric identities. In *proceedings of Mendel*, volume 98, page 4th.

Soderstrand, M. A. (2012). The moving interface between digital and analog circuits and its effect on future wireless systems. In *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*, pages 845–848.

Stoica, A., Zebulum, R., Guo, X., Keymeulen, D., Ferguson, M., and Duong, V. (2004). Taking evolutionary circuit design from experimentation to implementation: some useful techniques and a silicon demonstration. *Computers and Digital Techniques, IEE Proceedings* -, 151(4):295 – 300.

Tlelo-Cuautle, E. and Duarte-Villaseñor, M. (2008). Evolutionary electronics: Automatic synthesis of analog circuits by gas. In Yang, A., Shan, Y., and Bui, L.,

editors, *Success in Evolutionary Computation*, volume 92 of *Studies in Computational Intelligence*, pages 165–187. Springer Berlin / Heidelberg.

Tlelo-Cuautle, E., Guerra-Gómez, I., Duarte-Villaseñor, M., de La Fraga, L., Flores-Becerra, G., Reyes-Salgado, G., Reyes-García, C., and Rodriguez-Gómez, G. (2010). Applications of evolutionary algorithms in the design automation of analog integrated circuits. *Journal of Applied Sciences*, 10:1859–1872.

Varios (2013). Asco (a spice circuit optimizer).

Wang, F., Li, Y., Li, L., and Li, K. (2007). Automated analog circuit design using two-layer genetic programming. *Applied Mathematics and Computation*, 185(2):1087 – 1097. Special Issue on Intelligent Computing Theory and Methodology.

Yan, X. and Jin, J. (2010). Electronic circuits automatic design algorithm. In *Natural Computation (ICNC), 2010 Sixth International Conference on*, volume 5, pages 2334 –2337.

Yan, X., Wei, W., Liu, R., Zeng, S., and Kang, L. (2006). Designing electronic circuits by means of gene expression programming. In *Adaptive Hardware and Systems, 2006. AHS 2006. First NASA/ESA Conference on*, pages 194–199.

Zebulum, R., Pacheco, M., and Vellasco, M. (1998). Comparison of different evolutionary methodologies applied to electronic filter design. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 434 –439.

Zebulum, R. S., Vellasco, M. S., and Pacheco, M. A. (2000). Variable length representation in evolutionary electronics. *Evolutionary Computation*, 8(1):93–120.

Nomenclatura

A_v	Ganancia de voltaje
f_{low}	Frecuencia de corte inferior
Z_{in}	Impedancia de entrada
Z_{out}	Impedancia de salida
AGE	Analog genetic encoding
ASIC	Application specific integrated circuit, circuito integrado para aplicaciones específicas
AST	Abstract Syntax Tree, o árbol de sintaxis abstracta
CAD	Computer aided design, diseño asistido por ordenador
CPU	Central processing unit, unidad de proceso central
DSP	Digital signal processor, procesador digital de señal
EDA	Electronic aided design, diseño electrónico automatizado
EHW	Evolvable hardware, hardware evolutivo
FPGA	Field Programmable Gate Array, circuito que implementa una matriz de bloques lógicos cuyas conexiones pueden ser programadas. Puede ser reprogramada.
GE	Grammatical Evolution
GRN	Genetic regulatory network, red de regulación genética
HDL	Hardware description language, lenguaje descriptivo del hardware
IC	Integrated circuit, circuito integrado
MBF	Mean best fitness, o valor medio de las mejores adaptaciones
PG	Programación genética

RPC	Remote procedure call, o llamada a procedimiento remoto
SPICE	Simulation program with integrated circuits emphasis, programa de simulación con énfasis en circuitos integrados
SR	Success rate, o índice de éxitos
THD	Total harmonic distortion, distorsión armónica