Un DTD SGML per la generazione di codice NGSpice e CircuiTikZ

Renato Battistin

Sommario

La simulazione di un circuito elettronico e la sua rappresentazione grafica sono solitamente due operazioni distinte. Un DTD SGML che allo stesso tempo definisce i componenti dei circuiti, le loro connessioni, i comandi di simulazione e la rappresentazione grafica del circuito, consente di ottenere da un unico file SGML i codici NGSpice e CircuiTikZ.

Abstract

The simulation of an electronic circuit and its graphic representation are usually two distinct operations. An SGML DTD that simultaneously defines the circuit components, their connections, simulation commands, and graphical representation of the circuit, allows of to obtain the NGSpice and CircuiTikZ codes from a single SGML file.

1 Introduzione

La descrizione di un circuito elettronico e la sua simulazione è un problema affrontato dagli inizi dell'era informatica. I software sono innumerevoli e vanno dai primi simulatori per *mainframe* degli anni Settanta, come ad esempio *SPICE* di Laurence Nagel (Nagel), scritto ancora in Fortran, fino a quelli attuali, tra i quali possiamo annoverare

- **gSpiceUI**, motori di simulazione: ngSpice, GNU-Cap;
- **XCircuit**, schemi circuitali;
- **gEDA**, schemi circuitali in funzione PCB;
- **PartSim**, motore di simulazione e schema circuitale via interfaccia web;

solo per citarne alcuni, ma l'elenco esaustivo sarebbe lungo.

Questi simulatori hanno come scopo principale l'analisi del circuito che avviene dopo una descrizione, o meglio un'elencazione, dei componenti circuitali e delle loro connessioni elettriche. La rappresentazione grafica è un aspetto importante ma non strettamente necessario; come tale a volte è parte integrante del software di simulazione, a volte è derogata a programmi satellite. A seconda dei casi, l'utente può avere o meno la possibilità, più o meno ampia, di gestire la rappresentazione grafica, sia per i simboli circuitali che per gli altri elementi grafici come le etichette e le informazioni tecniche e gestionali del circuito elettronico.

Se guardiamo agli ambiti di impiego di questo tipo di software di simulazione, allora oltre a quello professionale è molto rilevante quello amatoriale, ma anche quello scolastico dove però sono richieste allo studente oltre alla conoscenza della materia elettronica, anche abilità di programmazione ed un minimo di capacità organizzativa per gestire la rappresentazione grafica circuitale.

Sia la programmazione della simulazione che la rappresentazione grafica hanno le loro specifiche esigenze. Ad esempio la descrizione circuitale al fine della simulazione generalmente richiede solamente una descrizione topologica dei componenti circuitali e non una loro descrizione topografica; in sostanza quello che importa sapere è a quali componenti elettronici è connesso un certo componente circuitale; più raramente se quei componenti sono fisicamente vicini o se il percorso di connessione è più o meno lungo[1]. Un simulatore circuitale richiede che vengano impartiti dei comandi appositi per eseguire una simulazione del comportamento fisico del circuito. Questi comandi dipendono dal tipo di simulazione richiesta: analisi dei piccoli segnali, transienti, analisi di Fourier, ecc.

Per contro la descrizione topografica del circuito elettronico richiede la rappresentazione delle connessioni fisiche oltre che quella dei singoli componenti circuitali; inoltre, affiché questa rappresentazione sia efficace, generalmente sono richiesti un certo numero di altri elementi grafici e tra questi senz'altro le etichette. Infine una serie di altre informazioni relative al circuito, ad esempio relative al suo impiego, alla sua progettazione, alla sua implementazione pratica, ecc., possono essere necessarie per completare la rappresentazione grafica del circuito elettronico.

In generale quindi i software di simulazione disponibili considerano la rappresentazione grafica un elemento, se non secondario, al più funzionale alla simulazione e seppure la resa grafica possa essere soddisfacente, ciò può non essere sufficiente in alcuni ambiti come ad esempio quello accademico.

Tra i pacchetti per il disegno tipografico dei circuiti elettronici ci sono:

1. Ovviamente dal punto di vista fisico la lunghezza di una connessione elettrica è importante in quanto può comportare ad esempio una capacità elettrica distribuita non trascurabile.

- la libreria `pst-circ` di PSTricks (Voss)
- il pacchetto MetaPost `mpcirc` di Tomasz J. Cholewo
- il pacchetto CircuiTikZ creato da Massimo A. Redaelli
- le macro M4 di Dwight Aplevich (Aplevich)

Questo articolo prende in considerazione solo CircuiTikZ per mostrare come un linguaggio di marcatura possa essere utilizzato per generare sia codice NGSpice, che codice integrabile direttamente in documenti *TEX. A tal scopo viene presentato un DTD SGML che consente la stesura in linguaggio marcato di codice che contestualmente ai comandi di simulazione permette di associare comandi per la generazione di codice CircuiTikZ direttamente utilizzabile in ambiente LATEX. Nel contempo la parte di codice SGML relativa alla simulazione può essere mappata in codice NGSpice, un dialetto *open source* molto diffuso di *Spice*, un linguaggio storico di eccellenza per la simulazione di circuiti elettronici.

Le sezioni seguenti mostrano le caratteristiche salienti di CircuiTikZ e di NGSpice, la loro organizzazione e resa nel DTD e le mappature adottate per traslitterare il codice SGML nei due linguaggi NGSpice e CircuiTikZ.

2 NGSpice

NGSpice, è un simulatore di circuiti elettronici conforme alle specifiche del linguaggio Spice. Il termine 'simulazione' ha un'accezione piuttosto ampia in quanto ammette varianti sulla base del problema circuitale studiato. Si va dalla semplice analisi delle condizioni stazionarie, ossia quello che in gergo elettronico viene detto 'punto di lavoro' del circuito; all'esame dei segnali transitori; all'analisi cosiddetta dei *piccoli segnali*;[2]; fino all'analisi termica, di Fourier, di MonteCarlo, del rumore e della sensibilità (*sensitivity*). Poiché software come NGSpice hanno la propria origine negli anni 70 del secolo scorso, non è inusuale trovarsi davanti a delle scarse capacità di rappresentazione tipografica dei risultati, in opposizione alle capacità di simulazione. Infatti spesso l'interazione con il software avviene mediante una *shell* testuale ed anche le moderne varianti di Spice normalmente si adeguano a questa impostazione. Se un'uscita grafica esiste, allora questa è sovente, come nel caso di NGSpice, solo una finestra *pop-up* che rappresenta l'andamento dei segnali ai nodi prescelti mentre non vi è alcuna rappresentazione grafica circuitale. Questo non significa che altre varianti di Spice non ce l'abbiano, anzi le più moderne varianti permettono di esaminare i segnali ai nodi a volte semplicemente usando il puntatore sulla rappresentazione grafica circuitale. Tuttavia il lavoro di costruzione circuitale in queste varianti di Spice moderne è comunque spesso delegato all'utente per via grafica, senza un corrispettivo strumento testuale alternativo, certamente molto più apprezzato, ad esempio, in ambito professionale.

Un codice NGSpice consiste in una serie di righe[3] che si distinguono fondamentalmente in due tipi: descrittore circuitale e direttiva di controllo. Le righe descrittrici generalmente sono costituite da un identificatore del componente circuitale, ad esempio un resistore, un transistor, un alimentatore; da una sequenza di nodi in base al numero di poli associati al componente elettronico, ad esempio due per un condensatore, tre per un transistor, ecc.; da un'eventuale stringa identificatrice del modello fisico del componente circuitale, come ad esempio nel caso di un diodo o, di nuovo, di un transistor; da un'eventuale successione di coppie di assegnazione parametro-valore, come ad esempio la temperatura di un resistore.

Le direttive di controllo sono contraddistinte da un punto, '.', come carattere iniziale; includono la prima riga di comando che obbligatoriamente deve essere quella del titolo, '.title'; l'ultima riga, '.end', che obbligatoriamente deve essere sempre riportata. Altre direttive di controllo hanno la funzione di indicare il tipo di modello, '.model', di un componente circuitale ove richiesto; l'esecuzione di un tipo di simulazione, ad esempio un transitorio, '.tran'; un tipo di analisi circuitale, ad esempio del rumore, '.noise'; la richiesta di un'uscita dati, ad esempio un grafico, '.plot'.

3 CircuiTikZ

Il pacchetto CircuiTikZ è utilizzabile per disegnare circuiti elettronici e fu scritto originariamente da Massimo Redaelli nel 2007 come strumento per la creazione di esercizi ed esami in ambito accademico; è ora mantenuto dall'autore e da altri due sviluppatori, Stefan Lindner e Stefan Erhardt (Redaelli *et al.*, 2017). Il pacchetto CircuiTikZ è compatibile con LATEX e ConTEXt. Le sue radici sono ben evidenti nella sua sintassi che sostanzialmente è quella di Ti*k*Z. Non è però del tutto compatibile con la libreria circuitale di Ti*k*Z (Tantau, 2015); l'uso contemporaneo dei due pacchetti richiede che venga specificata l'opzione `compatibility` di CircuiTikZ.

2. Un piccolo segnale è una variazione del segnale sufficientemente piccola da consentire un modello lineare per tutti i componenti del circuito (l'autore ringrazia Claudio Beccari per questa chiara e efficace definizione). Un piccolo segnale può essere aggiunto ad un segnale finito (ad esempio quello corrispondente al punto di lavoro), eventualmente in forma di segnale periodico e in seguito viene simulato il comportamento del circuito alla sua variazione, solitamente analizzando la risposta in frequenza ed in ampiezza.

3. Detta in gergo *netlist*.

Gran parte del circuito elettronico viene disegnato con CircuiTikZ percorrendo la rete circuitale per collocare lungo il percorso i nodi e tra di essi le connessioni mediante delle linee, oppure dei componenti elettronici[4]. Attributi dei nodi e delle connessioni completano le informazioni quali le etichette, le loro posizioni, la forma delle connessioni, aspetti grafici come colore ed evidenziazione, ecc.

4 Sintassi

Le sintassi di NGSpice e di CircuiTikZ sono piuttosto diverse tra loro. La prima è incentrata sulla descrizione del componente elettronico richiedendo la dichiarazione esplicita di tutti i suoi nodi oltre che della sua natura fisica:

```
componente1 nodo1 nodo2 ...
componente2 nodo2 nodo3 ...
...
```

La seconda, essendo incentrata sulla descrizione del percorso, può essere ridotta alla forma minimale

```
nodo1 componente1 nodo2 componente2 ...
```

dove in quest'ultimo caso il vocabolo `componente` include anche componenti grafici che non hanno un corrispettivo componente elettronico in NGSpice come, ad esempio, la semplice linea di collegamento tra due nodi e il nodo di terra.

I vincoli maggiori per impostare una sorta di sintassi comune tra i due linguaggi sono presentati da NGSpice, che richiede obbligatoriamente che ciascun componente sia descritto su una, ed una sola, riga di codice. Per contro CircuiTikZ, ereditando la versatilità della sintassi di Ti*k*Z, può accettare del codice sia distribuito su più righe, sia ridondante, ragion per cui possiamo adattare la sintassi di CircuiTikZ alla forma seguente[5]:

```
nodo1 componente nodo2 nodo2 \
componente2 nodo3 nodo3 ...
```

Questa sintassi verrà quindi sfruttata nel DTD per definire i componenti circuitali in modo tale che il medesimo codice SGML sia traslitterabile in entrambe le sintassi NGSpice e CircuiTikZ.

5 Il DTD

5.1 Scopo

Il DTD è rivolto principalmente alla simulazione. Questo significa che la rappresentazione grafica è opzionale per tutti i componenti circuitali. Questa scelta è dettata dalla considerazione che la simulazione è un processo generalmente volto al miglioramento delle prestazioni circuitali e, come tale, modifica progressivamente il numero ed il tipo di componenti circuitali impiegati, ed anche il tipo ed il numero di simulazioni richieste; solo al termine di questo percorso sorge eventualmente l'esigenza di ottenere una rappresentazione grafica. Questa forma del DTD può penalizzare leggermente il suo impiego in ambito scolastico nel caso in cui la rappresentazione grafica del circuito sia l'unica esigenza, in questo caso può essere sufficiente impiegare direttamente CircuiTikZ.

5.2 Traslitterazione

La traslitterazione del codice SGML in codice NGSpice oppure CircuiTikZ viene eseguita mediante dei file qui chiamati di *mappatura*. Questi file hanno una sintassi conforme ai file di sostituzione dell'*Amsterdam SGML Parser*[6] e sono forniti come argomento al software `sgmlsaps` per traslitterare il suo input standard generato a sua volta dal *parser* `nsgmls`, ad esempio[7]:

```
cat doc.sgml | nsgmls -c catalog | \
sgmlsasp map > doc.tex
```

dove `doc.sgml` è il file SGML da traslitterare, `map` è il file di mappatura, `doc.tex` è il file LaTeX generato; il file `catalog` come dice il nome è un cosiddetto file di catalogo che generalmente viene fornito come opzione a `nsgmls` per includere nella traslitterazione alcuni file SGML standard come ad esempio un file di entità SGML predefinite.

5.3 Organizzazione

Il DTD, essendo orientato principalmente alla simulazione, presenta una struttura essenzialmente consona, con un titolo, una descrizione topologica del circuito e quindi un'analisi ed una stampa dei risultati:

```
<!ELEMENT ngspice - o (title, circuit,
    control?, analysis?,
print-results?, end-line)
    +(includefile|comment|tikz)>
```

In pratica attualmente solo la parte di codice SGML che descrive il circuito, delimitata dall'elemento[8] `<circuit>`, può produrre codice CircuiTikZ mentre le altre parti del codice sono essenzialmente funzionali alla simulazione e quindi vengono mappate esclusivamente in codice NGSpice. Tuttavia il DTD prevede la possibilità di includere direttamente del codice LaTeX e Ti*k*Z a qualsiasi livello del documento SGML:

4. Con un meccanismo che ricorda la *turtle graphics* del linguaggio di programmazione Logo.

5. Il carattere '\' ha il classico significato di indicare che il codice seguente continua in effetti sulla medesima riga.

6. https://web.cs.wpi.edu/~kal/elecdoc/sgml/ASPhead.html.

7. Per un esempio applicativo si confrontino i sempre utili *Appunti di Informatica Libera* di Daniele Giacomini, http://wwwcdf.pd.infn.it/AppuntiLinux/a2558.htm.

8. Con un minimo di abuso di linguaggio identifichiamo l'elemento con il suo `tag` di apertura.

FIGURA 1: Il classico simbolo di un amplificatore operazionale assieme ad un'etichetta ottenuta mediante inclusione di codice Ti*k*Z.

```
<!NOTATION TikZ system "">
<!ELEMENT tikz - - CDATA >
<!ATTLIST tikz format NOTATION (Tikz)
    "Tikz">
```

In tal caso il codice viene gestito come una notazione[9] e quindi al di fuori dell'ambito di validazione SGML. Ad esempio il codice SGML

```
<!doctype ngspice system 'ngspice.dtd'>
<ngspice>
  <title>Testo accanto a componente
      circuitale
   gestito via notazione 'TikZ'</title>
  <circuit>
    <opamp name="myopamp" node="(2,2)">
    <right-angle-circuit>
      <xy-minus>(myopamp.out)</xy-minus>
      <!-- one of (HVLINE VHLINE
          STRAIGHT) -->
      <straight>
      <xy-plus>(3,2)</xy-plus>
    </right-angle-circuit>
    <tikz>(3.5,1.75) rectangle +(2.5,
        0.5) +(1.25, 0.25)
 node {\tiny \textbf Amplificatore
     Operazionale}</tikz>
  </circuit>
  <end-line>
</ngspice>
```

viene mappato al sequente codice CircuiTikZ

```
%
% Testo accanto a componente circuitale
    gestito via notazione 'TikZ'
%
\begin{circuitikz}[scale=1.2]\draw[gray]
(2,2) node [op amp] (myopamp) {}
(myopamp.out) -- (3,2)(3.5,1.75)
    rectangle +(2.5, 0.5) +(1.25, 0.25)
 node {\tiny \textbf Amplificatore
     Operazionale}
;\end{circuitikz}
```

che può essere incluso direttamente nel proprio documento LaTeX e compilato (Fig. 1).

5.4 Componenti circuitali ed elementi grafici

L'elemento `<circuit>`:

```
<!ELEMENT circuit - -
    (%ngspice-elements; |
    %circuitikz-elements; |
    %tikzgraphics; | subcircuit |
    model)+ >
```

attualmente permette di inserire una serie di componenti circuitali NGSpice:

```
<!ENTITY % ngspice-elements
    'xspace-code-model |
    behavioral-source | capacitor |
    diode |
    linear-voltage-controlled-current |
    linear-voltage-controlled-voltage |
    linear-current-controlled-current |
    linear-current-controlled-voltage |
    jfet | coupled-inductors | inductor
    | mosfet | num-dev-gss |
    lossy-trams-line | bjt | resistor |
    switch | single-lossy-trasm-line |
    mesfet | voltage | current'>
```

intercalati eventualmente al loro modello e ad eventuali sotto-circuiti. Inoltre consente l'inserimento di alcuni elementi "grafici" di CircuiTikZ

```
<!ENTITY % circuitikz-elements 'opamp |
    block-diagram-component |
    short-circuit | right-angle-circuit
    | open-circuit'>
```

che ovviamente non essendo componenti circuitali non vengono mappati a codice NGSpice.
L'elemento `<block-diagram-component>` rappresenta uno qualsiasi dei diagrammi a blocchi previsti da CircuiTikZ, escluso l'amplificatore operazionale al quale è riservato l'elemento apposito `<opamp>`:

```
<!ELEMENT block-diagram-component - o
    EMPTY >
<!ATTLIST block-diagram-component
        name cdata #required
        node cdata #required
        symbol cdata #required
        label cdata #implied
        label-position cdata #implied
        more-options cdata #implied >
```

dove la distinzione viene fatta mediante l'attributo `symbol` all'atto della mappatura:

```
<block-diagram-component> + "[NODE]␣node␣
    \[[SYMBOL]\]␣([NAME])␣
    {[MORE-OPTIONS]}"
</block-diagram-component>
```

5.5 Sezioni marcate nel DTD

La separazione mediante l'impiego di entità parametriche tra componenti circuitali da una parte e modello e sotto-circuito dall'altra è utile quando si desidera utilizzare il DTD solo per rappresentare schemi circuitali e non per la simulazione. Implementando infatti nel DTD una coppia di *sezioni marcate*,[10]

```
<!-- SGML variant for mapping to
    CircuiTikz -->
<![ %ngspice; [ <!ELEMENT ngspice - o
    (title, circuit, control?,
    analysis?, print-results?, end-line)
    +(includefile & comment & latex &
    tikz)> ]]>
```

9. La parola chiave `NOTATION` in sostanza afferma che il testo delimitato dell'elemento <tikz> non viene sottoposto alle usuali regole di *parsing* ma viene passato tale e quale, all'atto della mappatura del codice SGML, al fantomatico programma 'Ti*k*Z' che se ne prenderà auspicabilmente carico.

10. Le sezioni marcate hanno una funzione simile ad un condizionale e consentono di includere o escludere alcune sezioni del DTD per cambiarne la destinazione d'uso.

```
<!-- SGML variant for mapping to NGSpice
    -->
<![ %circuitikz; [ <!ELEMENT ngspice - o
    (title, circuit) +(latex & tikz)> ]]>
```

è possibile attivare la sezione del DTD relativa alla sola descrizione circuitale, essenzialmente l'unica ad essere mappata a codice CircuiTikZ come detto in precedenza:

```
<!doctype ngspice system 'ngspice.dtd' [
<!ENTITY % ngspice "IGNORE">
<!ENTITY % circuitikz "INCLUDE">
]>
```

5.6 Rappresentazione circuitale in CircuiTikZ

CircuiTikZ può rappresentare un intero circuito elettronico mediante un unico comando `draw` ed in effetti la mappatura dell'elemento `<circuit>` segue questa impostazione:

```
<circuit> +
    "\\begin\{circuitikz\}\[[OPTIONS]\]
\\draw"
</circuit> + ";\\end{circuitikz}" +
```

Tuttavia in alcuni casi, per esempio se si vuole evidenziare una parte del circuito con un colore oppure una grafica differente, può essere utile inserire nell'ambiente `circuitikz` ulteriori comandi `draw`. Il DTD attualmente supporta i comandi `draw` e `filldraw`, ciascuno con la possibilità di introdurre tutte le opzioni desiderate:

```
<!ENTITY % tikzgraphics 'draw | filldraw
    | node-labels' >
<!ELEMENT (%tikzgraphics;) - o EMPTY >
<!ATTLIST (%tikzgraphics;)
        options cdata #implied >
```

I due elementi, una volta mappati, chiudono il comando precedente con un carattere ';' ed iniziano quello nuovo, nuove opzioni incluse:

```
<draw> ";␣\n\\draw\[[OPTIONS]\]␣"
</draw>
<filldraw> ";␣\n\\filldraw\[[OPTIONS]\]␣"
</filldraw>
```

Ad esempio consideriamo il codice SGML di un circuito per il raddoppio della tensione di alimentazione (Roberts e Sedra, 1997, p. 97) mostrato in Fig. 3:

```
<!doctype ngspice system 'ngspice.dtd'>
<ngspice>
  <title>Voltage Doubler - (Roberts
      &amp;amp; Sedra, Spice, 2nd Ed.,
      Oxford)</title>
  <circuit>
    <voltage name="in" node-plus="1"
        node-minus="0" xy-plus="(0,2)"
        xy-minus="(0,0)" symbol="sV"
        label="10V" to-options="-*">
      <sinusoidal vo="0" va="10"
          freq="1kHz">
    </voltage>
    <short-circuit xy-minus="(0,0)"
        xy-plus="(2,0)">
    <capacitor name="1" node-plus="2"
        node-minus="1" symbol="eC"
        value="1u" xy-minus="(0,2)"
        xy-plus="(2,2)" label="$C_1␣
        (1\micro\farad)$"
        label-position="_"
        to-options="*-">
    </capacitor>
    <diode name="1" node-plus="0"
        node-minus="2" model="DIN4148"
        label="$D_1$" xy-plus="(2,0)"
        xy-minus="(2,2)">
    </diode>
    <ground xy-node="(2,0)">
    <draw options="dashed">
    <diode name="2" node-plus="2"
        node-minus="3" model="DIN4148"
        label="$D_2$" label-position="^"
        xy-plus="(2,2)" xy-minus="(4,2)">
    </diode>
    <capacitor name="2" node-plus="3"
        node-minus="0" symbol="eC"
        value="1u" xy-minus="(4,0)"
        xy-plus="(4,2)" label="$C_2␣
        (1\micro\farad)$"
        label-position="_">
    </capacitor>
    <short-circuit xy-plus="(4,0)"
        xy-minus="(2,0)">
    <draw>
    <short-circuit xy-plus="(5,0)"
        xy-minus="(4,0)">
    <short-circuit xy-plus="(5,2)"
        xy-minus="(4,2)">
    <open-circuit xy-plus="(5,2)"
        xy-minus="(5,0)"
        to-options="o-o">
    <node-labels
        options="(0,2)node[anchor=south]␣
        {1}␣(2,2)node[anchor=south]{2}␣
        (5,2)node[anchor=south]␣{3}">
    <model mname="DIN4148" type="D">
      <parameter pname="Is" pval="0.1pA">
      <parameter pname="Rs" pval="16">
      <parameter pname="CJO" pval="2p">
      <parameter pname="Tt" pval="12n">
      <parameter pname="Bv" pval="100">
      <parameter pname="Ibv" pval="0.1p">
    </model>
  </circuit>
  <analysis>
    <tran-analysis time-step="100u"
        time-stop="10m" time-start="0m"
        time-step-maximum="100u">
  </analysis>
  <end-line>
</ngspice>
```

La sua rappresentazione grafica di questo codice, una volta mappato, è mostrata in Fig. 2, dove la parte di circuito tratteggiata è stata introdotta da un comando `draw`; il codice CircuiTikZ è riportato qui di seguito:

```
%
% Voltage Doubler - (Roberts \& Sedra,
    Spice, 2nd Ed., Oxford)
%
\begin{circuitikz}[scale=1.2,american]
\draw(0,0) to[sV, l=10V, -* ] (0,2)
(0,0) to[short, l=,  ] (2,0)
(2,2) to[eC, l_=$C_1 (1\micro\farad)$,
    *- ] (0,2)
(2,0) to[Do, l=$D_1$,  ] (2,2)
(2,0) node[ground] {};
```

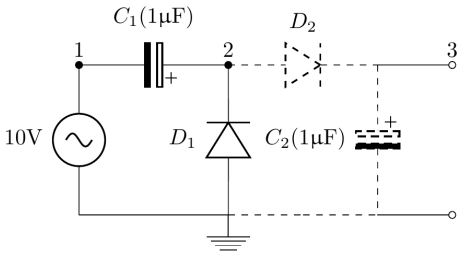

Figura 2: Immagine del raddoppiatore di tensione.

```
\draw[dashed]
(2,2) to[Do, l^=$D_2$,  ] (4,2)
(4,2) to[eC, l_=$C_2 (1\micro\farad)$,
    ] (4,0)
(2,0) to[short, l=,  ] (4,0);
\draw[]
(4,0) to[short, l=,  ] (5,0)
(4,2) to[short, l=,  ] (5,2)
(5,0) to[open, l=, o-o ] (5,2)
    {(0,2)node[anchor=south] {1}
    (2,2)node[anchor=south]{2}
    (5,2)node[anchor=south] {3}}
;\end{circuitikz}
```

Si noti che il condensatore polarizzato[11], così come il diodo, sono componenti bipolari che, a differenza di altri quali il resistore, ammettono un polo positivo ed uno negativo. Il disegno del componente bipolare "inizia" dal suo polo positivo e quindi la mappatura assume l'aspetto seguente:

```
<diode> + "[XY-PLUS]␣to\[[SYMBOL],␣
    l[LABEL-POSITION]=[LABEL],␣
    [TO-OPTIONS]␣\]␣[XY-MINUS]"
</diode>
...
<capacitor> + "[XY-PLUS]␣to\[[SYMBOL],␣
    l[LABEL-POSITION]=[LABEL],␣
    [TO-OPTIONS]␣\]␣[XY-MINUS]"
</capacitor>
```

dove apparentemente sembra che due componenti elettronici fisicamente distinti vengano mappati in maniera identica. Questo ovviamente non è vero, in quanto il valore dell'attributo `SYMBOL` viene gestito in forma predefinita entro il DTD al momento della definizione degli attributi dell'elemento rappresentante il componente elettronico, ad esempio:

```
<!ATTLIST diode
        name cdata #required
        node-plus cdata #required
        node-minus cdata #required
        model cdata #required
        symbol cdata "Do"
        label cdata #implied
        label-position cdata #implied
        xy-plus cdata #implied
        xy-minus cdata #implied
        to-options cdata #implied >
```

Le etichette dei tre nodi principali del circuito sono inserite come codice CircuiTikZ prima della chiusura dell'ambiente `circuitikz`; il codice viene generato mediante mappatura dell'elemento `<node-labels>`:

```
<node-labels> "␣{[OPTIONS]}"
</node-labels>
```

Gli elementi SGML per così dire 'comuni' sia a CircuiTikZ che a NGSpice sono stati definiti nel DTD in modo tale che i loro attributi siano nell'ordine prima tutti quelli caratteristici di NGSpice, come gli identificatori alfanumerici dei poli di connessione ed il valore della grandezza fisica caratterizzante il componente elettronico, ad esempio 50kΩ; poi tutti gli attributi caratteristici di CircuiTikZ e, più in generale, di Ti*k*Z come le coppie ordinate cartesiane dei poli circuitali, le etichette, la loro posizione, il tipo di connessione, ecc.:

```
<capacitor name="2" node-plus="3"
    node-minus="0" value="1u"
    xy-plus="(4,2)" xy-minus="(4,0)"
    label="$C_2$">
```

Come accennato in precedenza, essendo il DTD principalmente orientato a NGSpice, solo gli attributi relativi a quest'ultimo sono obbligatori:

```
<!ELEMENT capacitor - o  (par-model?,
    par-ac?, par-m?, par-scale?,
    par-temp?, par-dtemp?, par-tc1?,
    par-tc2?, ic?)>
<!ATTLIST capacitor
        name cdata #required
        node-plus cdata #required
        node-minus cdata #required
        symbol cdata "C" -- circuital
            symbol --
        value cdata #required --
            capacitance value --
        xy-plus cdata #implied
        xy-minus cdata #implied
        label cdata #implied
        label-position cdata #implied
        to-options cdata #implied >
```

Un medesimo sorgente SGML può essere mappato sia a codice CircuiTikZ, che a codice NGSpice; questo, una volta eseguito, permette di ottenere, ad esempio, l'andamento transitorio iniziale della tensione al nodo di uscita '3' confermando il raddoppio della tensione di alimentazione (Fig. 3).

5.7 Unità di misura

Il pacchetto CircuiTikZ integra il pacchetto `siunitx`[12]; tuttavia è possibile impiegare anche il pacchetto `SIunits` ad esempio direttamente entro l'attributo `label` dell'elemento `capacitor` come si può vedere nel codice SGML del raddoppiatore di tensione.

5.8 Elementi multipolari

Il codice più sotto riportato rappresenta un circuito elettronico mostrato in Fig. 4 comprendente anche un elemento multipolare, un transistor:

11. Fisicamente può essere un condensatore elettrolitico.

12. Anche se nelle ultime versioni è disabilitato nelle impostazioni predefinite.

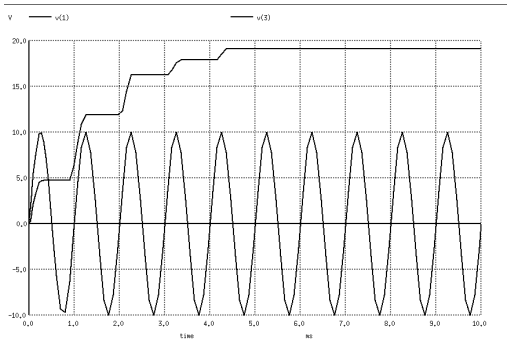

Figura 3: Transitorio iniziale della tensione di uscita al nodo '3' del circuito raddoppiatore di tensione così come mostrato da NGSpice assieme all'andamento della tensione d'ingresso al nodo '1'.

```
<!doctype ngspice system 'ngspice.dtd'>
<ngspice>
  <title>Voltage Gain of a Transistor
      Amplifier</title>
  <circuit>
    <ground xy-node="(0,-0.5)">
    <comment>DC supply</comment>
    <voltage name="BB" node-plus="5"
        node-minus="0" xy-plus="(0,0.5)"
        xy-minus="(0,-0.5)"
        symbol="battery1" label="$V_{BB}␣
        (3\volt)$" label-position="_">
      <dc-tran value="3V">
    </voltage>
    <comment>Small signals
        source</comment>
    <voltage name="i" node-plus="4"
        node-minus="5" xy-plus="(0,2)"
        xy-minus="(0,0.5)" symbol="sV"
        label="$v_i␣(1\milli\volt)$"
        to-options="*-*">
      <ac acmag="1mV">
    </voltage>
     <short-circuit xy-plus="(0,2.5)"
         xy-minus="(0,2)">
    <resistor name="B" node-plus="3"
        node-minus="4" value="100K"
        xy-plus="(2,2.5)"
        xy-minus="(0,2.5)"
        label="$R_{BB}␣(100\kilo\ohm)$">
    </resistor>
    <bjt name="1" collector-node="2"
        base-node="3" emitter-node="0"
        model="npn_transistor"
        xy-node="(2.5,2.5)"
        id-node="bjt1" >
    </bjt>
    <short-circuit xy-plus="(bjt1.base)"
        xy-minus="(2,2.5)"
        to-options="*-">
    <short-circuit
        xy-plus="(bjt1.emitter)"
        xy-minus="(2.5,2)"
        to-options="-*">
    <model mname="npn_transistor"
        type="npn">
      <parameter pname="Is"
          pval="1.8104e-15">
      <parameter pname="Bf" pval="100">
    </model>
    <ground xy-node="(2.5,2)">
    <resistor name="C" node-plus="1"
        node-minus="2" value="3K"
        xy-plus="(2.5,4.5)"
        xy-minus="(bjt1.collector)"
        label="$R_C␣(3\kilo\ohm)$"
        to-options="*-">
    </resistor>
    <voltage name="CC" node-plus="10"
        node-minus="1" xy-plus="(2.5,6)"
        xy-minus="(2.5,4.5)"
        label="$V_{CC}␣(10\volt)$"
        to-options="*-">
      <dc-tran value="10V">
    </voltage>
    <short-circuit xy-plus="(2.5,6)"
        xy-minus="(3.5,6)"
        to-options="-*">
    <ground xy-node="(3.5,6)">
    <node-label node="(2.5,6)" label="1"
        anchor="east">
    <node-label node="(bjt1.collector)"
        label="2" anchor="east">
    <node-label node="(bjt1.collector)"
        label="$v_o$" anchor="west">
    <node-label node="(2.5,2)" label="0"
        anchor="east">
    <node-label node="(2,2.5)" label="3"
        anchor="south">
    <node-label node="(0,2)" label="4"
        anchor="east">
    <node-label node="(0,0.5)" label="5"
        anchor="east">
    <comment>transistor collector source
        polarity symbols</comment>
    <node-label node="(2.5,5.25)"
        label="-" anchor="east">
    <node-label node="(2.5,5.25)"
        label="+" anchor="east">
    <comment>transistor base source
        polarity symbols</comment>
    <node-label node="(0,0)" label="+"
        anchor="east">
    <node-label node="(0,0)" label="-"
        anchor="east">
  </circuit>
  <end-line>
</ngspice>
```

Un transistor è rappresentato in CircuiTikZ mediante un singolo nodo dotato però di tre oppure quattro connettori,[13] quindi per identificarne la posizione è sufficiente una singola coppia di coordinate cartesiane:

```
<bjt name="1" collector-node="2"
    base-node="3" emitter-node="0"
    model="npn_transistor"
    xy-node="(2.5,2.5)" id-node="bjt1" >
```

I connettori invece sono individuati tramite dei suffissi che richiamano la loro funzione; ad esempio, se il nodo rappresentante un transistor è identificato da CircuiTikZ con la stringa `bjt1`, allora il riferimento alla sua base è dato dalla stringa `bjt1.base`.

Sempre nel codice SGML precedente si osservi l'impiego dell'elemento `<comment>` di ovvia interpretazione e come viene mappato nel codice CircuiTikZ riportato più sotto; si tratta comuque di una singola linea di commento ed in tale forma

13. Un transistor ammette a volte un quarto connettore che viene utilizzato per collegare a terra il cosiddetto *case* ossia l'involucro.

viene mappata anche nel codice NGSpice. Un'altra osservazione riguarda i segni di polarità delle sorgenti di tensione costante che sono stati apposti *ad-hoc* anche se a livello normativo può essere tranquillamente usato il corrispodente simbolo senza i segni. Si noti inoltre al termine del codice SGML la sequenza di istanze dell'elemento `<node-label>` a completamento delle informazioni e dell'aspetto del circuito elettronico. Ecco il codice CircuiTikZ:

```
%
% Voltage Gain of a Transistor Amplifier
%
\begin{circuitikz}[scale=1.2,american]
\draw(0,-0.5) node[ground] {}
% DC supply
(0,-0.5) to[battery1, l=$V_{BB}
    (3\volt)$,  ] (0,0.5)
% Small signals source
(0,0.5) to[sV, l=$v_i (1\milli\volt)$,
    *-* ] (0,2)
(0,2) to[short, l=,  ] (0,2.5)
(0,2.5) to[R, l=$R_{BB} (100\kilo\ohm)$,
     ] (2,2.5)
(2.5,2.5) node[npn] (bjt1) {}
(2,2.5) to[short, l=, *- ] (bjt1.base)
(2.5,2) to[short, l=, -* ] (bjt1.emitter)
(2.5,2) node[ground] {}
(bjt1.collector) to[R, l=$R_C
    (3\kilo\ohm)$, *- ] (2.5,4.5)
% (2.5,4.5) to[V, l=$V_{CC} (+10\volt)$,
    -o ] (2.5,6)
(2.5,6) to[battery1, l_=$V_{CC}
    (10\volt)$, *- ] (2.5,4.5)
(2.5,6) to[short, l=,] (3.5,6)
(3.5,6) node[ground] {}
(2.5,6) node[anchor=east] {1}
(bjt1.collector) node[anchor=east] {2}
(bjt1.collector) node[anchor=west]
    {$v_o$}
(2.5,2) node[anchor=east] {0}
(2,2.5) node[anchor=south] {3}
(0,2) node[anchor=east] {4}
(0,0.5) node[anchor=east] {5}
% transistor collector source polarity
    symbols
(2.5,5.25) node[anchor=south east] {-}
(2.5,5.25) node[anchor=north east] {+}
% transistor base source polarity symbols
(0,0) node[anchor=south east] {+}
(0,0) node[anchor=north east] {-}
;\end{circuitikz}
```

5.9 Componenti non-NGSpice

Esiste un'intera classe di componenti elettronici non previsti in forma nativa in NGSpice ma che ciononostante vengono rappresentati in CircuiTikZ, come le porte logiche, i doppi dipoli (es. il trasformatore), ecc. In NGSpice sono composti di, o riconducibili a, componenti elettronici elementari che possono essere resi disponibili mediante librerie. In CircuiTikZ questi componenti sono identificati come entità e quindi nel DTD non hanno un elemento dedicato. Il DTD mette a disposizione un elemento generico `<node-component>` che può essere impiegato per definire volta per volta il componente desiderato, come ad esempio una porta logica oppure un filtro passa-basso, ed inol-

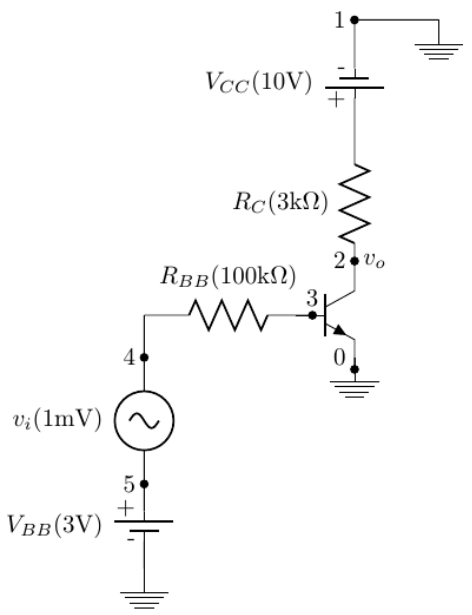

Figura 4: L'immagine dell'amplificatore di tensione a singolo stadio.

tre anche un elemento generico `<node-label>` per etichettare i nodi:

```
<!doctype ngspice system 'ngspice.dtd'>
<ngspice>
  <title>Logic gates</title>
  <circuit>
    <node-component xy-node="(0,2)"
        component="and␣port"
        id-node="myand1">
    <node-component xy-node="(0,0)"
        component="and␣port"
        id-node="myand2">
    <node-component xy-node="(2,1)"
        component="xnor␣port"
        id-node="myxnor">
    <right-angle-circuit>
      <xy-plus>(myxnor.in 1)</xy-plus>
      <hvline>
      <xy-minus>(myand1.out)</xy-minus>
    </right-angle-circuit>
    <right-angle-circuit>
      <xy-plus>(myxnor.in 2)</xy-plus>
      <hvline>
      <xy-minus>(myand2.out)</xy-minus>
    </right-angle-circuit>
    <node-label id-node="myand1"
        sub-id-node="out" anchor="south␣
        west" label="AND␣1␣out">
    <node-label id-node="myand2"
        sub-id-node="out" anchor="north␣
        west" label="AND␣2␣out">
    <node-label id-node="myxnor"
        sub-id-node="out" anchor="west"
        label="XNOR␣out">
  </circuit>
  <end-line>
</ngspice>
```

Il codice SGML una volta mappato

```
%
% Logic gates
```

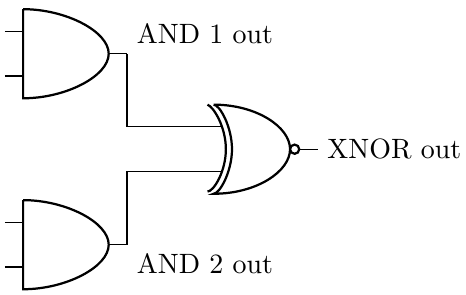

Figura 5: Porte logiche.

```
%
\begin{circuitikz}[scale=1.2,american]
\draw(0,2) node[and port] (myand1) {}
(0,0) node[and port] (myand2) {}
(2,1) node[xnor port] (myxnor) {}
(myxnor.in 1) -| (myand1.out)
(myxnor.in 2) -| (myand2.out)
(myand1.out) node[anchor=south west]
    {AND 1 out}
(myand2.out) node[anchor=north west]
    {AND 2 out}
(myxnor.out) node[anchor=west] {XNOR out}
;\end{circuitikz}
```

e compilato, fornisce l'immagine in Fig. 5.

6 Conclusioni

L'attuale DTD è da considerarsi ancora in fase di sviluppo seppure la struttura generale delle parti finora implementate possa definirsi stabile. La parte tipografica attualmente supporta la maggioranza dei componenti elettronici non-NGSpice di CircuiTikZ in forma generica, come visto nel caso delle porte logiche. Questo comporta inevitabilmente un minore supporto in fase di stesura del codice SGML per l'utente, il quale deve ricorrere alle proprie conoscenze di CircuiTikZ, e può eventualmente dare origine a problemi di validazione. Il DTD attuale è anche privo di riferimenti a comandi CircuiTikZ quali `ctikzset` che consentirebbe di adeguare parametricamente la rappresentazione grafica dei componenti elettronici; l'ambiente `scope`, seppure concettualmente semplice da implementare; i tripoli come il tiristore, il potenziometro e il deviatore; anche i cosiddetti *percorsi di transistor* (*transistor paths*) non sono stati ancora implementati. Tutti questi componenti e comandi possono attualmente essere descritti in SGML, ma solo scrivendo direttamente il codice CircuiTikZ protetto entro l'elemento `<tikz>`. Complessivamente si può affermare che, seppure utilizzabile, il DTD attuale richiede comunque una conoscenza piuttosto approfondita di CircuiTikZ e in qualche misura anche di Ti*k*Z, soprattutto per quanto concerne le sue regole di disegno.

7 Ringraziamenti

L'autore desidera ringraziare Claudio Beccari per le sue preziose osservazioni relative ai circuiti elettronici citati nell'articolo.

Riferimenti bibliografici

Aplevich, D. *M4 Macros for Electric Circuit Diagrams in LaTeX Documents*. URL http://ctan.mirror.garr.it/mirrors/CTAN/graphics/circuit_macros/doc/CMman.pdf.

Nagel, L. W. «The origins of spice». URL http://www.omega-enterprises.net/The%20Origins%20of%20SPICE.html.

Redaelli, M. A., Lindner, S. e Erhardt, S. (2017). *CircuiTikZ*. URL http://ctan.mirror.garr.it/mirrors/CTAN/graphics/pgf/contrib/circuitikz/doc/circuitikzmanual.pdf.

Roberts, G. W. e Sedra, A. S. (1997). *SPICE*. Oxford University Press.

Tantau, T. (2015). *The Ti*k*Z and PGF Packages Manual*. Institut fur Theoretische Informatik Universitat zu Lubeck. URL http://ctan.mirror.garr.it/mirrors/CTAN/graphics/pgf/base/doc/pgfmanual.pdf.

Voss, H. *pst-circ - A PSTricks package for drawing electric circuits*. URL http://ctan.mirror.garr.it/mirrors/CTAN/graphics/pstricks/contrib/pst-circ/doc/pst-circ-doc.pdf.

▷ Renato Battistin
rbattistin at apf dot it