Síntese de Alto Nível utilizando Programação Linear Inteira

Paulo Flores

INESC/IST
Instituto de Engenharia de Sistemas e Computadores
Rua Alves Redol, 9 - 1000 Lisboa - Portugal
Tel: +351.1.3100000 Fax: +351.1.3145843
E-mail: pff@inesc.pt

Maio 1997

Resumo

Neste relatório apresentam-se sumariamente três modelos de síntese de alto nível utilizando programação linear inteira. Descreve-se de forma breve o programa desenvolvido para a manipulação do grafo de fluxo de dados e a geração uma formulação ILP (Integer Linaer Programming) satisfazendo esses modelos. Um exemplo de geração e resolução da formulação ILP com vários modelos é apresentado utilizando uma descrição teste (benchmark). Finalmente compara-se para todos as descrições teste de síntese de alto nível, e como os vários modelos descritos, os tempos de resolução das ILP com diferentes programas.

Índice

1 Síntese de alto nível

A geração automática de descrições RTL a partir de descrições de alto nível é designada de síntese de alto nível.

Este processo é geralmente dividido em diversos sub-problemas mais simples de resolver [1]. Estes sub-problemas são:

- planeamento temporal (*scheduling*)
- alocação de recursos (*allocation*)
- selecções de recursos
- atribuição (*biding*) de recursos
- alocação de multiplexers
- alocação de registos

A forte interdependência entre o planeamento e alocação de recursos permite que a estrutura do circuito possa ser obtida quer se comece por uma ou outra tarefa. É mesmo possível entrelaçar as duas tarefas, com o objectivo de reduzir ao mínimo uma determinada função de custo, que dependerá do número de passos de controlo e dos recursos de *hardware* necessários.

Tradicionalmente, as ferramentas desenvolvidas para síntese de alto nível resolvem cada um destes sub-problemas separadamente, usando mesmo em alguns casos algoritmos heurísticos. No entanto, devido a estes sub-problemas estarem fortemente inter-relacionados as soluções encontradas podem não ser óptimas, resultando assim em circuitos de qualidade inferior.

A optimização do planeamento temporal (*scheduling*) e alocação de recursos (*allocation*) na síntese de alto nível pode-se dividir em 4 categorias de acordo com o objectivo a optimizar [2]:

- UCS - *Unconstrained scheduling* - minimiza uma função dependente do número de recursos utilizados e do número de passos de controlo necessários.
- RCS - *Resource-constrainted scheduling* - minimiza o número de passo de controlo quando os recursos a utilizar estão fixos.
- TCS - *Time-constrained scheduling* - minimiza o número de recursos quando o números de passos de controlo está fixo.
- TRCS - *Time-and-resource-constrained scheduling* - optimiza uma dada função quando o número de passos de controlo e recursos a utilizar estão ambos fixos.

A última categoria também é designada de FCS (*Feasible-constrained scheduling*) [3], ou seja, dado um número de passos de controlo e de recursos a utilizar, verifica-se se estes satisfazem o problema de síntese de alto nível. Se satisfizeram, determina-se qual é a solução óptima para os recursos e passos de controlo dados.

1.1 Técnicas Básicas de planeamento

A técnica mais básica de planeamento é denominada de ASAP (*as soon as possible*) Esta técnica consiste em atribuir em cada passo de controlo as operações cujos predecessores já tiverem acabado de computar. Este processo é repetido até que todas as operações tenham sido atribuídas a um dado passo de controlo. Assim, cada operação é realizadas imediatamente depois de ter todos os seus operandos disponíveis, ou seja o mais cedo possível.

A técnica de planeamento ALAP (*as late as possible*) é de certa forma a "dual" da técnica ASAP. As operações vão sendo atribuídas aos passos de controlo o mais tarde possível. Começando-se no último passo de controlo para o primeiro cada operação é atribuída a um dado passo se todos os seus sucessores já tiverem um passo de controlo definido.

Outras técnicas básicas de planeamento podem ser encontradas em [3] e [4]

2 Formulação ILP

A formulação do problema de síntese de alto nível em programação linear inteira, ILP (*integer linear programming*), permite encontrar uma solução óptima satisfazendo simultaneamente o conjunto de restrições impostos por cada um dos sub-problemas.

A programação linear inteira é um modelo matemático concebido para determinar um conjunto de valores inteiros que, optimizando uma determinada função objectivo, satisfaz ao mesmo tempo um conjunto de restrições lineares. Usando a notação matricial um problema de programação linear inteira pode ser descrito como:

$$\begin{array}{ll} \text{optimizar} & c \cdot x \\ \text{restringido a} & A \cdot x \geq b \ , \\ & x \geq 0 \end{array}$$

sendo x o vector dos valores inteiros a determinar, A a matriz de restrições, b e c vectores de coeficientes.

A síntese de alto nível recorrendo a uma fórmulas IlP é realizada em três passos:

1. Obtenção do grafo de fluxo de dados, DFG (*data flow graph*), e de fluxo de controlo, CFG (*controlo flow graph*).

2. Manipulação do grafo para para obter a formulação ILP desejada satisfazendo um conjunto de restrições impostas pelo projectista.

3. Resolução do problema ILP gerado de forma a obter uma solução óptima para a síntese de alto nível.

Na figura 1 apresentam-se uma selecção possível para as ferramentas usadas em cada um dos passo. Para a geração dos CFG e DFG é utilizada a ferramenta `Sir/Castele` desenvolvida no GMD *Gesellschaft fűr Mathematik und Datenverarbeitung* [5]. Esta ferramenta aceita com descrições da alto nível nas linguagens `C, C++, VHDL` e `Verilog`. A geração da formulação ILP é feita usado a uma ferramenta desenvolvida no INESC/IST cuja descrição com mais detalhe é feita na secção 3. A resolução de cada uma das ILPs geradas é feita recorrendo a um programa de domínio público que resolve problemas de programação linear, `lp_solve`, [6].

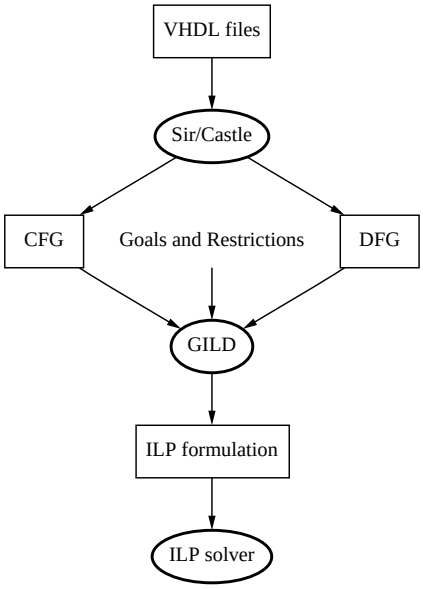

Figura 1: Síntese de alto nível com formulação ILP

A notação usada para descrever a formulação do problema de síntese de alto nível em programação linear inteira (ILP) é a seguinte [3]:

$DFG(V, E)$ **:** representa o grafo de fluxo de dados, constituído por n operações ($n = |V|$, sendo V o conjunto de nós do grafo), e dependências de dados ($e = |E|$, sendo E o conjunto de arcos do grafo) que vai ser planeado em s passos de controlo.

o_i **:** representa uma operação realizada por um dado nó ($1 \leq i \leq n$).

$o_i \rightarrow o_j$ **:** implica uma realção de precedência entre a operação o_i e o_j, onde o_j é o sucessor imediato de o_i.

S_i **ou** t_i^S **:** representa o menor tempo de planeamento possível para a operação o_i, geralmente obtido através do algoritmo ASAP.

L_i **ou** t_i^L **:** representa o maior tempo de planeamento possível para a operação o_i, geralmente obtido através do algoritmo ALAP.

t_i **:** indica o passo de controlo em que é iniciada a operação o_i.

FU_{t_k} **:** unidade funcional do tipo t_k. Se esta unidade pode realizar a operação o_i então diz-se que $o_i \in FU_{t_k}$. O custo da unidade funcional do tipo t_k é dado por c_{t_k}, existindo m tipos de unidades disponíveis.

$x_{i,j}$ **:** é uma variável binária (variável inteira restringida a $0 \leq x_{i,j} \leq 1$) associada à operação o_i definida da seguinte forma:

$$x_{i,j} = \begin{cases} 1, & \text{se a operação } o_i \text{ for planeada para o passo de controlo } j \\ 0, & \text{se a operação } o_i \text{ for planeada outro passo de controlo} \end{cases}$$

2.1 Modelo básico - optimização da recursos

O modelo básico para a formulação ILP considera que cada unidade funcional realiza a sua operação num único passo de controlo. Este modelo é constituído por um conjunto de três restrições:

1. Cada operação é apenas iniciada uma única vez. Ou seja, cada operação têm um único passo de controlo associado na solução:

$$\sum_{j=S_i}^{L_i} x_{i,j} = 1, \qquad \text{para } 1 \leq i \leq n; \tag{1}$$

2. A sequência de relações entre as operações representadas pelo grafo $DFG(V, E)$ tem que ser mantida no planeamento. Isto significa que para cada arco $o_i \rightarrow o_j$ temos que ter $t_j \geq t_i + 1$, Sabendo que t_i se pode exprimir em função das variáveis $x_{i,j}$, através da expressão $t_i = \sum_{j=L_i}^{S_i} j \cdot x_{i,j}$, então as restrições impostas pelas dependências do grafo podem ser formuladas da seguinte forma:

$$t_i - t_j \leq 1, \qquad \text{para todos os arcos } o_i \rightarrow o_j$$

   ou seja,

$$\sum_{j=L_i}^{S_i} j \cdot x_{i,j} - \sum_{j=L_i}^{S_i} j \cdot x_{i,j} \leq 1 \qquad \text{para todos os arcos } o_i \rightarrow o_j \tag{2}$$

3. Finalmente o número total de unidades funcionais usadas em cada passo está limitada a M_{t_k}. Os valores de M_{t_k} podem ser determinados pela resolução do problema ILP ou serem impostos pelo projectista na própria formulação.

$$\sum_{o_i \in FU_{t_k}} x_{i,j} - M_{t_k} \leq 0 \qquad \text{para } 1 \leq j \leq s;\ 1 \leq k \leq m; \tag{3}$$

Para minimizar dos recursos, a função objectivo a optimizar será dada por:

$$\text{minimizar} \quad \sum_{k=1}^{m} (c_{t_k} \cdot M_{t_k}) \tag{4}$$

2.2 Modelo multi-ciclo

Neste modelo considera-se que as operações o_i podem podem ser executadas em unidades funcionais FUt_k que necessitam de mais que um passo de controlo. Assim, se designarmos por d_i o número de passos de controlo necessários para execução da operação o_i, temos respeitar que $t_i - t_j \leq d_i$ para todos os arcos $o_i \rightarrow o_j$. Em termos de programação ILP temos que alterar a equação (2) para:

$$\sum_{j=L_i}^{S_i} j \cdot x_{i,j} - \sum_{j=L_i}^{S_i} j \cdot x_{i,j} \leq d_i \qquad \text{para todos os arcos } o_i \rightarrow o_j \tag{5}$$

A contabilização das unidades funcionais utilizadas em cada passo de controlo têm agora que ter em conta que cada operação pode levar vários passos de controlo a ser executada. Portanto a equação (3) é substituída por:

$$\sum_{p=0}^{d_k-1} \sum_{o_i \in FU_{t_k}} x_{i,j-p} - M_{t_k} \leq 0 \qquad \text{para } 1 \leq j \leq s;\ 1 \leq k \leq m; \tag{6}$$

2.3 Modelo com optimização de registos

A minimização do número de registos corresponde à minimização do tempo de vida útil das variáveis envolvidos no DFG. Este tempo define-se como o intervalo entre o passo de controlo em que uma variável é produzida e último passo de controlo em ela é utilizada. Para cada operação o_i o tempo de vida útil da sua variável de saída é dado por $SLK_i = \max_{o_i \rightarrow o_j}(t_j - t_i - d_i)$ para todos os arcos de saída de o_i.

Na formulação ILP, a minimização do tempo de vida útil das variáveis é conseguido pela introdução de um conjunto extras de restrições, que calculam os valores de SLK_i,

$$t_j - t_i - d_i - SKL_i \leq 0 \qquad \text{para todos } o_i \rightarrow o_j \tag{7}$$

e pela modificação da função objectivo por forma a minimizar estes valores,

$$\text{minimizar} \quad p_1 \cdot \sum_{k=1}^{m} (c_{t_k} \cdot M_{t_k}) + p_2 \cdot \sum_{i=1}^{n} SLK_i \tag{8}$$

onde p_1 e p_2 são pesos que se pretende dar a cada um dos objectivos de minimização, recursos utilizados e número de registos, respectivamente.

3 Ferramenta de manipulação de grafos e geração de ILP.

A ferramenta desenvolvida para manipulação de grafos e geração de formulações ILP, conforme os modelos apresentados na secção anterior, designa-se por **GILD** - *Graph manipulation and ILP generation for High-Level Synthesis*.

A presente versão da ferramenta GILD (Version 2.2.87) suporta o conjunto de comandos apresentados na tabela 1.

User and interface commands	
`?`	Synonym for 'help'.
`bug`	To help debuging the program.
`cd`	Change to directory DIR.
`echo`	Echo command.
`graph`	List/select avaible graph.
`help`	Display this text.
`z`	List files in dir (ls -FClg).
`pwd`	Print the current working directory.
`quit`	Quit program.
`set`	Set/show values of variables.
`sh`	Command for shell (system).
`source`	Read commands from a file.
Input/Output commands	
`read_graph`	Read a graph from FILE.
`write_graph`	Write graph description [to a file].
`write_ilp`	Write ILP formualtion [to a file].
`write_sol`	Write ILP solution [to a file].
Graph manipulation commands	
`classify`	Classify all nodes of a graph.
`force_seq`	Add edges to force operation sequencing.
`rm_trans`	Remove all (or N level) transitive edges.
`rm_unlink`	Delete unlinked nodes.
`serial`	Serialize all nodes of a DFG.
`unlink_nops`	Unlink nodes which have no operations.
Scheduling commands	
`init_sched`	Initilaize asap/alap sched values.
`alap`	Calculate ALAP scheduling for all nodes.
`asap`	Calculate ASAP scheduling for all nodes.
`solve`	Find the ILP solution using lp_solve.
Analysis commands	
`info`	Print info about all nodes in tha graph.
`view_graph`	View graph using graphviz edDG (system).
`view_sol`	View solution graph.

Tabela 1: Lista de comandos do GILD

Na figura 2 apresenta-se o conjunto de programas externos necessários à execução de alguns comandos e o interface por eles utilizados.

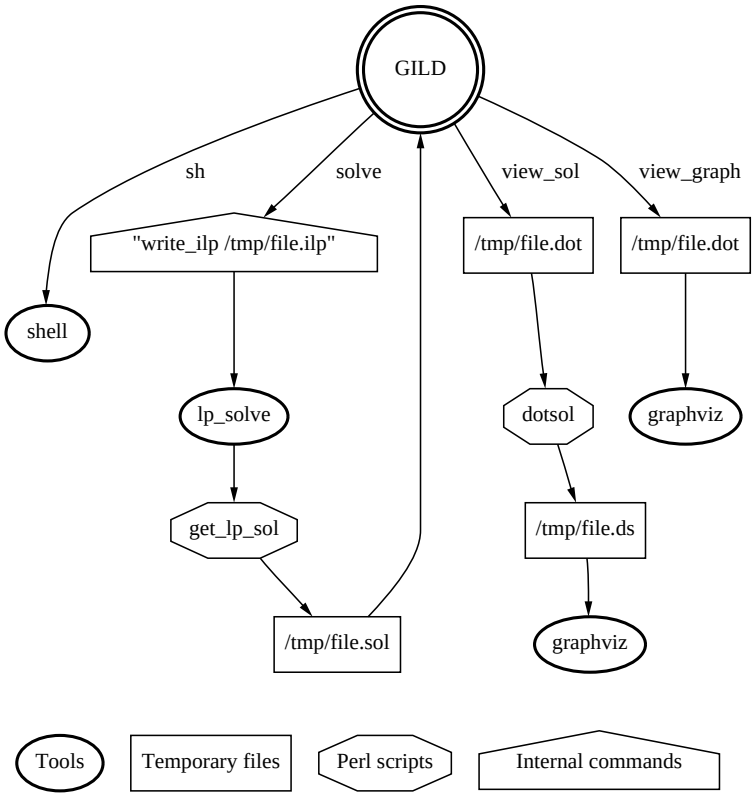

Figura 2: Interface dos comandos externos à ferramenta GILD

O tipo de modelo ILP considerado para o problema de síntese de alto nível é controlado no programa GILD através de variáveis. Nesta versão são suportadas as variáveis apresentadas na tabela 2. Estas permitem fazer a selecção entre os vários modelos apresentados na secção anterior.

Variable	**Def. Value**	**Description**
`echo`	0	Echo each command before execution
`bin`	0	Assume that all ILP variables are binary
`01`	1	Write [0,1] restrictions for variables
`int`	1	Write integer restrictions for variables
`multicycle`	0	Consider multicyle operations
`min_res`	1	Minimize resources area (weight in obj.)
`min_reg`	0	Minimize registers number (weight in obj.)

Tabela 2: Variáveis para controlo do GILD

As variáveis `bin, 01` e `int` "não alteram" a formulação ILP do problema de síntese de alto nível, "apenas" facilitam a interface da ferramenta GILD para diferentes tipos de programas de resolução de ILPs.

A variável `multicyle` permite considerar que certas unidades funcionais realizam as suas operações em mais que um passo de controlo. Assim, quando esta variável está a 1 é considerado que as unidades funcionais correspondente aos operadores `*` e `/` realizam as suas função em 3 passos de controlo (na versão 2.2.87). Nesta situação é usada a formulação ILP apresentada na secção 2.2.

As variáveis `min_res` e `min_reg` permitem dar pesos na função objectivo da ILP à áreas dos recursos utilizados e ao número de ciclos que as variáveis têm que ser guardadas em registos, respectivamente. Se a variável `min_reg` for diferente de 0 então será acrescentado à formulação do problema as equações apresentadas na secção 2.3.

3.1 Exemplo de síntese de alto nível

Nesta secção apresenta-se os resultados na obtidos para os sub-problemas de planeamento temporal (*schduling*) e alocação de recursos (*allocation*) utilizando a ferramenta GILD.

O circuito seleccionado para exemplo é um dos que constituiu o conjunto de *benchmarks* de síntese de alto nível, `diffeq`. Resultados obtidos com os outros circuitos podem ser encontrados na secção 4.

Na figura 3 apresenta-se a descrição do circuito em VHDL do algoritmo para a resolução em *hardware* da equação diferencial $\frac{\partial^2 y}{\partial x} + 3x\frac{\partial y}{\partial x} + 3y = 0$, utilizando cálculos com inteiros de virgula fixa.

Nesta descrição, u representa $\frac{\partial y}{\partial x}$ e considera-se que ∂x é aproximado por $x_{i+1} - x_i$ (tal como $y_{i+1} - y_i$ aproxima ∂y e $u_{i+1} - u_i$ aproxima ∂u)[1] O valor de a determina o número de vezes que o ciclo é executado para o cálculo da solução. A variáveis u_{i+1}, x_{i+1} e y_{i+1} representam o novos valores de u, x e y, repectivamente. Assim, tem-se que $x_{i+1} = x_i + dx$, $y_{i+1} = u\partial x_i + y_i$ e $u_{i+1} = u_i - 3x_i u_i \partial x_i - 3y_i \partial x_i$. A descrição pressupõe que os valores iniciais de x, y, u, dx, e a são carregados antes de cada cálculo.

A através da ferramente `Sir/Castel` obetêm-se os grafos de controlo de dados (CFG) e de fluxo de dados (DFG). O grafo da figura 4 representa o grafo de controlo. Cada nó de conrolo tem associado um grafo de fluxo de dados, por exemplo os nós 0 e 2 têm associados os sub-grafos que se iniciam nos nós 0 e 17 (`df:0` e `df:17`) do grafo de fluxo de dados. Estes grafos encontra-se nas figuras 5 e 6. As figuras 7, 8 e 9 representam os grafos de fluxo de dados associados aos outros nós de controlo, 1, 4 e 5, repectivamente.

Para a ferramenta GILD é lido um grafo que é a junção de todos grafos de fluxo de dados apresentados. É sobre esse grafo que são realizadas operações de simplificação para se chegar a um grafo que representa o problema de síntese de alto nível, o $DFG(V, E)$, e sobre o qual se extrarirá a fromulação ILP.

[1] Na descrição VHDL o valor de x_i é representado pela variável `x` e o valor de x_{i+1} por `x1`. Representações semelhantes são feitas para os valores de y, y_{i+1}, u e u_{i+1}

```
entity diffeq is
   port (Xinport: in integer;
         Xoutport: out integer;
         DXport: in integer;
         Aport: in integer;
         Yinport: in integer;
         Youtport: out integer;
         Uinport: in integer;
         Uoutport: out integer);
end diffeq;

architecture diffeq of diffeq is
begin
  P1: process (Aport, DXport, Xinport, Yinport, Uinport)
    variable x_var,y_var,u_var, a_var, dx_var: integer ;
    variable x1, y1, t1,t2,t3,t4,t5,t6: integer ;
  begin
    x_var := Xinport;
    a_var := Aport;
    dx_var := DXport;
    y_var := Yinport;
    u_var := Uinport;
    while (x_var < a_var) loop
      t1 := u_var * dx_var;
      t2 := 3 * x_var;
      t3 := 3 * y_var;
      t4 := t1 * t2;
      t5 := dx_var * t3;
      t6 := u_var - t4;

      u_var := t6 - t5;
      y1 := u_var * dx_var;
      y_var := y_var + y1;
      x_var := x_var + dx_var;
    end loop;
    Xoutport <= x_var;
    Youtport <= y_var;
    Uoutport <= u_var;
  end process P1;
end diffeq;
```

Figura 3: Descrição VHDL do circuito.

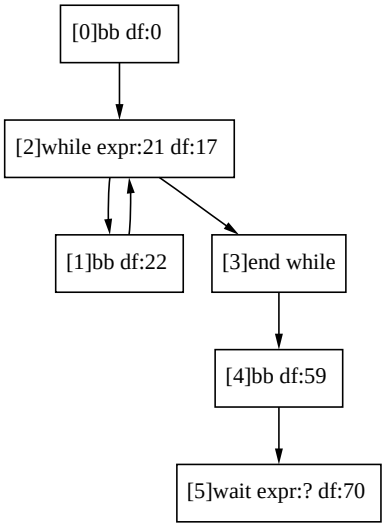

Figura 4: O grafo de fluxo de controlo

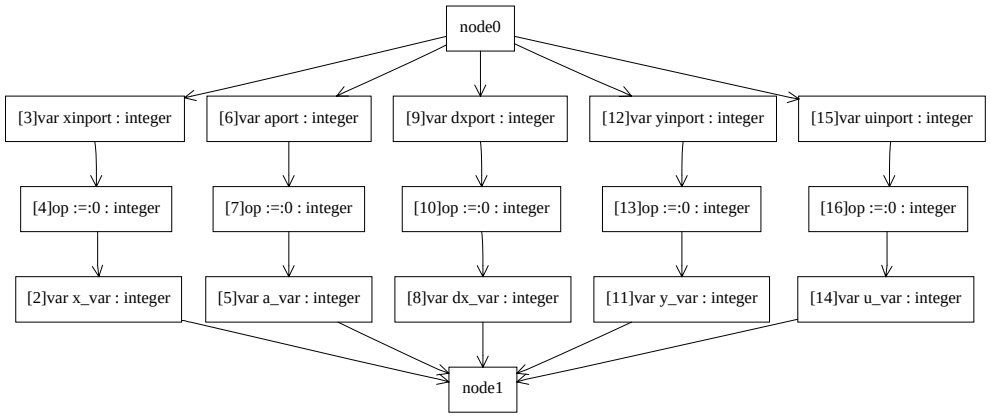

Figura 5: O grafo de fluxo de dados associado ao nó de controlo 0.

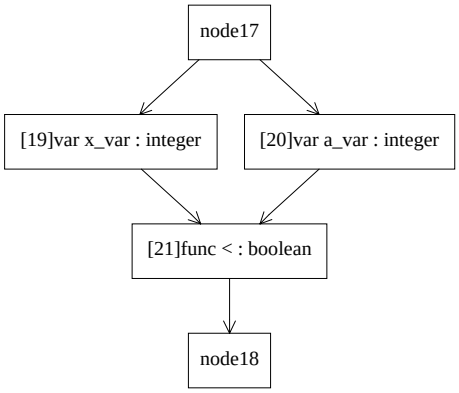

Figura 6: O grafo de fluxo de dados associado ao nó de controlo 2.

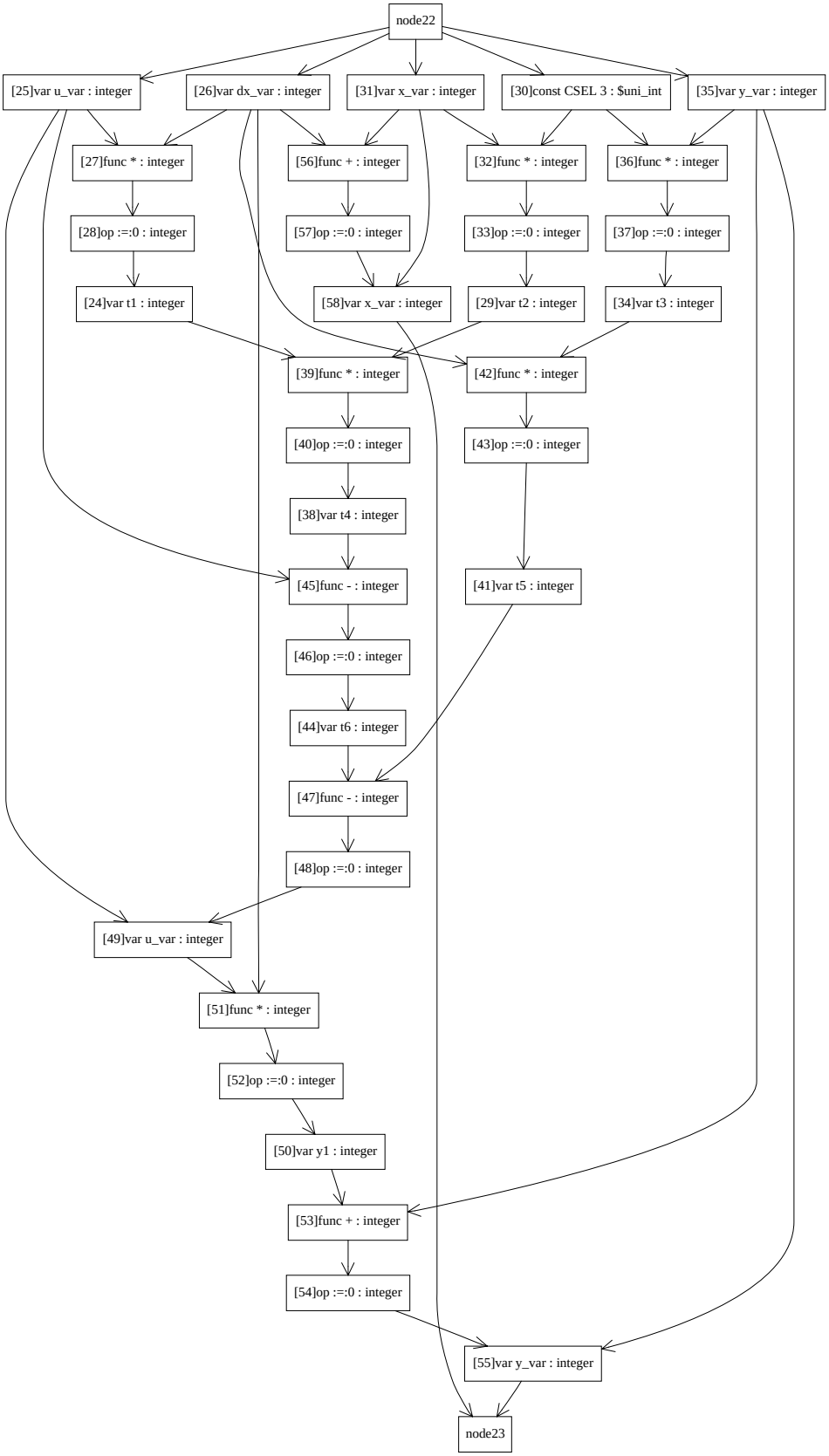

Figura 7: O grafo de fluxo de dados associado ao nó de controlo 1.

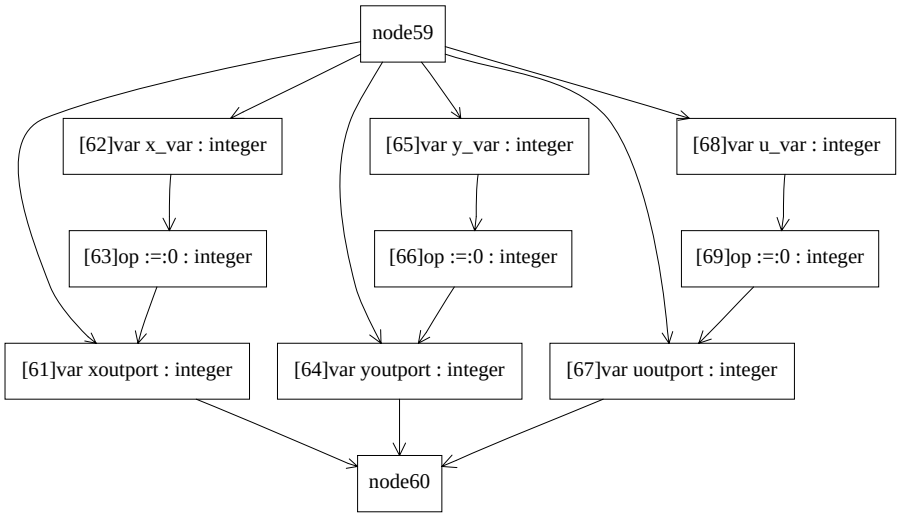

Figura 8: O grafo de fluxo de dados associado ao nó de controlo 4.

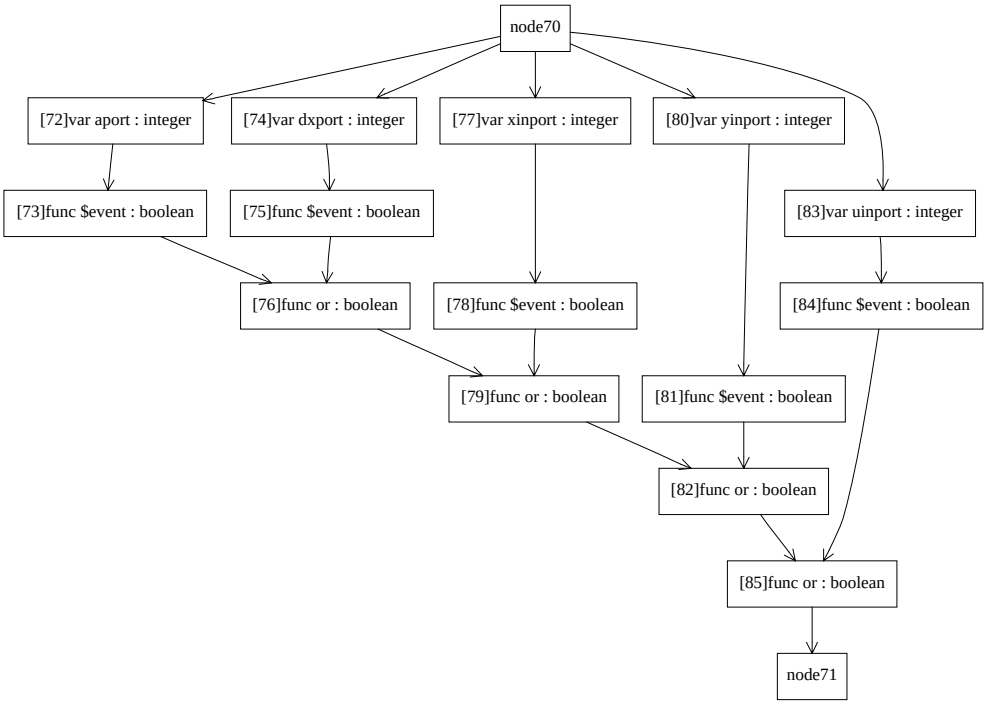

Figura 9: O grafo de fluxo de dados associado ao nó de controlo 5.

A sequência de comados realizada para obter o grafo simplificado apresentado na figura 10 foi a seguinte:

```
gild> read_graph diffeq.Dfg
gild> serial
                node1 -> node17
                node18 -> node22
                node23 -> node59
                node60 -> node70
gild> classify
gild> force_seq
                node27 ~> node47
                node45 ~> node47
                node32 ~> node56
                node36 ~> node53
gild> unlink_nops
gild> rm_unlink
gild> rm_trans all
```

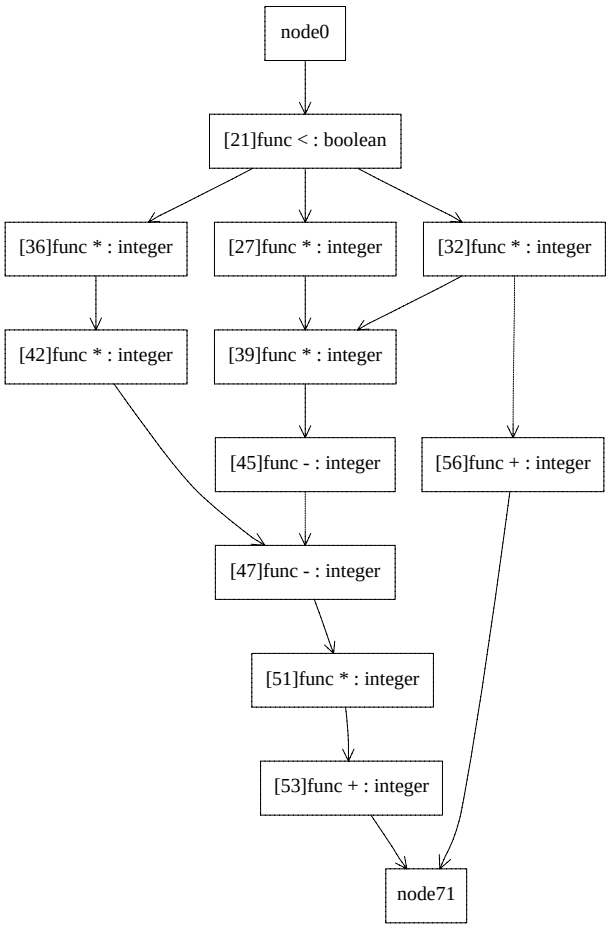

Figura 10: O grafo simplificado de fluxo de dados.

Para obter a formulação ILP do problema representado pelo grafo da figura 10 executaram-se os seguintes comandos:

```
gild> classify
gild> init_sched
gild> asap
gild> alap
```

A fórmulação ILP deste problema encontra-se na figura 11 e foi obtida através do comando `write_ilp`. Note-se que esta formulação corresponde à fromulação básica apresentada na secção 2.1.

```
/* Genrated automatically by gild converter (pff@inesc.pt)    */
/* gild, Version 2.2.87  compiled on May 16 1997 at 11:45:13  */
/* Copyright (c) 1997 -  Paulo Flores <pff@inesc.pt>          */
/* File : diffeq.Dfg                                          */
/* Graph: diffeq                                              */
/* Nodes: 13                                                  */
/* Edges: 15                                                  */
/* Sched: [0,  7]                                             */
/* Options set:                                               */
/*  echo    = 0  Echo each command before execution           */
/*  bin     = 0  Assume that all ILP variables are binary     */
/*  01      = 1  Write [0,1] restrictions for variables       */
/*  int     = 1  Write integer restrictions for variables     */
/*  multicycle = 0  Consider multicyle operations                */
/*  min_res = 1  Minimize resources area (weight in obj.)     */
/*  min_reg = 0  Minimize registers number (weight in obj.)   */

/* for resource minimization */
min: + 1 ADDs + 1 SUBs + 1 MULs + 1 LTs;

                            /*--- all operations must start only once ---*/

op1: + 1 Xnode21_0 = 1;
op2: + 1 Xnode27_1 = 1;
op3: + 1 Xnode32_1 = 1;
op4: + 1 Xnode36_1 + 1 Xnode36_2 = 1;
op5: + 1 Xnode39_2 = 1;
op6: + 1 Xnode42_2 + 1 Xnode42_3 = 1;
op7: + 1 Xnode45_3 = 1;
op8: + 1 Xnode47_4 = 1;
op9: + 1 Xnode51_5 = 1;
op10: + 1 Xnode53_6 = 1;
op11: + 1 Xnode56_2 + 1 Xnode56_3 + 1 Xnode56_4 + 1 Xnode56_5 + 1 Xnode56_6 = 1;
op12: + 1 Xnode0_0 = 1;
op13: + 1 Xnode71_7 = 1;

                        /*--- constraints based on the sequencing graph ---*/

si1: + 1 Xnode36_1 + 2 Xnode36_2 >= 1;
si2: + 2 Xnode56_2 + 3 Xnode56_3 + 4 Xnode56_4 + 5 Xnode56_5 + 6 Xnode56_6 - 1 Xnode32_1 >= 1;
si3: + 2 Xnode42_2 + 3 Xnode42_3 - 1 Xnode36_1 - 2 Xnode36_2 >= 1;
si4: + 4 Xnode47_4 - 2 Xnode42_2 - 3 Xnode42_3 >= 1;
si5: + 7 Xnode71_7 - 2 Xnode56_2 - 3 Xnode56_3 - 4 Xnode56_4 - 5 Xnode56_5 - 6 Xnode56_6 >= 1;

                                          /*--- resource constraints ---*/

rLTs0: + 1 Xnode21_0 - 1 LTs <= 0;
rMULs1: + 1 Xnode27_1 + 1 Xnode32_1 + 1 Xnode36_1 - 1 MULs <= 0;
rMULs2: + 1 Xnode36_2 + 1 Xnode42_2 - 1 MULs <= 0;
rADDs2: + 1 Xnode56_2 - 1 ADDs <= 0;
rMULs3: + 1 Xnode42_3 - 1 MULs <= 0;
rADDs3: + 1 Xnode56_3 - 1 ADDs <= 0;
```

```
rADDs4: + 1 Xnode56_4 - 1 ADDs <= 0;
rSUBs4: + 1 Xnode47_4 - 1 SUBs <= 0;
rMULs5: + 1 Xnode51_5 - 1 MULs <= 0;
rADDs5: + 1 Xnode56_5 - 1 ADDs <= 0;
rADDs6: + 1 Xnode53_6 + 1 Xnode56_6 - 1 ADDs <= 0;


                          /*--- bound variables to [0, 1] interval ---*/

b1: Xnode21_0 <= 1;
b2: Xnode0_0 <= 1;
b3: Xnode27_1 <= 1;
b4: Xnode32_1 <= 1;
b5: Xnode36_1 <= 1;
b6: Xnode36_2 <= 1;
b7: Xnode42_2 <= 1;
b8: Xnode56_2 <= 1;
b9: Xnode42_3 <= 1;
b10: Xnode56_3 <= 1;
b11: Xnode56_4 <= 1;
b12: Xnode47_4 <= 1;
b13: Xnode51_5 <= 1;
b14: Xnode56_5 <= 1;
b15: Xnode53_6 <= 1;
b16: Xnode56_6 <= 1;
b17: Xnode71_7 <= 1;
b18: ADDs >= 1;
b19: SUBs >= 1;
b20: MULs >= 1;
b21: LTs >= 1;

                            /*--- integer constraint on variables ---*/

int Xnode21_0;
int Xnode0_0;
int Xnode27_1;
int Xnode32_1;
int Xnode36_1;
int Xnode36_2;
int Xnode42_2;
int Xnode56_2;
int Xnode42_3;
int Xnode56_3;
int Xnode56_4;
int Xnode47_4;
int Xnode51_5;
int Xnode56_5;
int Xnode53_6;
int Xnode56_6;
int Xnode71_7;
int ADDs;
int SUBs;
int MULs;
int LTs;
```

Figura 11: Formulação ILP para o circuito `diffeq` com o modelo básico.

A solução deste problema obtido através do programa `lp_solve` encontra-se representado não figura 12, à qual corresponde a utilização dos seguintes recursos: 1 multiplicador, 1 comparador de *menor*, 1 subtrator e 2 multiplicadores. Com este recursos são necessários 7 passos de controlo para implementar em *hardware* o algoritmo descrito.

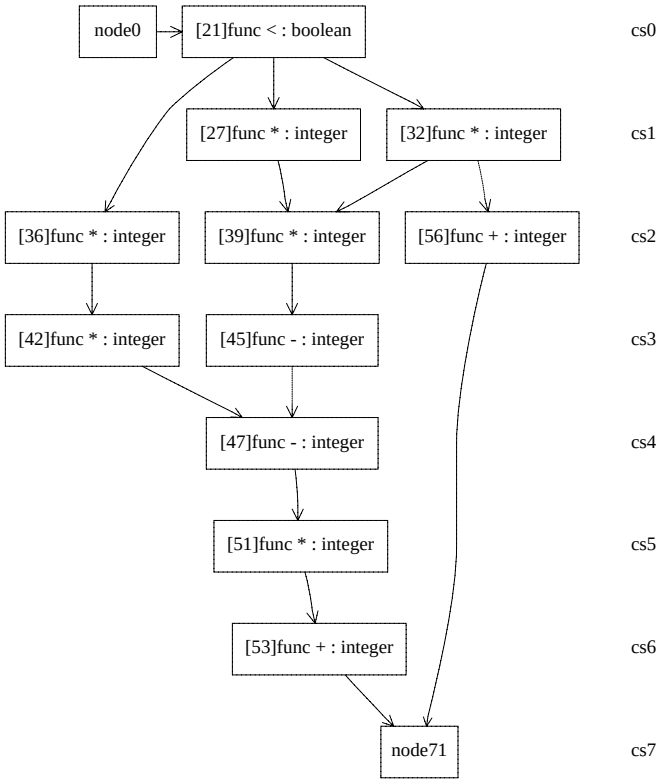

Figura 12: Solução para o circuito `diffeq` com modelo básico.

Na figura 13 apresenta-se a solução do mesmo problema, mas agora considerando que as operações de multiplicação são multi-ciclo e levam 3 passo de controlo para executar. Neste caso o número de recursos utilizados é igual ao obtido na solução anterior com excepção do número de multiplicadores que aumentou, sendo agora necessários 3 em vez de 2. Naturalmente, devido às operação de multiplicação necessitarem de mais passos de controlo, o número total de passos de controlo para implementar o algoritmo aumentou, sendo agora necessário 13 passos de controlo.

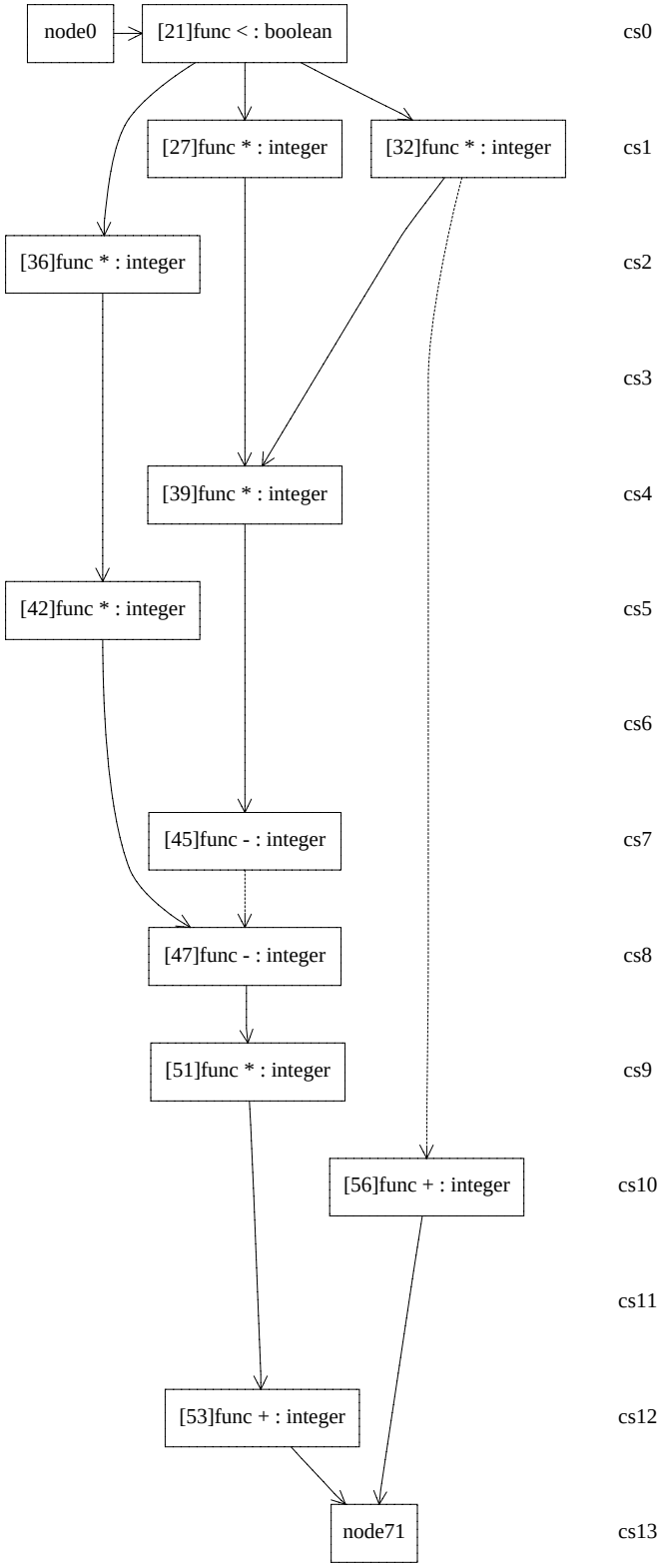

Figura 13: Solução para o circuito `diffeq` com modelo multi-ciclo.

4 Estudo comparativo - Modelos vs Solucionadores

A escolha do modelo a utilizar influencia não só o resultado obtido como tempo de resolução do problema de síntese de alto nível. Assim realizou-se um estudo comparativo do tempo de execução da vários solucionadores (*solvers*) sobre os modelos anteriormente descritos. O solucionadores, programas de resolução de problemas ILP, seleccionados foram:

Cplex - programa comercial para a resolução de problemas lineares em geral, [7].

lp_solve - programa de domínio público desenvolvido na universidade de Eindhoven para resolução de problemas lineares inteiros e mistos, [6].

opbdp - programa de domínio público desenvolvido na Alemanha, Max-Planck-Institut fuer Informatik, que utiliza um algoritmo de enumeração implícita para resolução de problemas lineares contendo apenas variáveis binárias, [8].

01solo - programa desenvolvido no INESC/IST que utiliza também um algoritmo de enumeração implícita (mas melhorado) para resolução de problemas lineares contendo apenas variáveis binárias.

Na tabela 3 apresentam-se o tempo de resolução da formulação ILP de síntese de alto nível para um conjunto de descrições teste (*benchmarks*), utilizando os diferentes programas de resolução de ILPs. Para cada descrição teste foram gerados os grafos de controlo e fluxo de dados através das ferramentas `Sir/Castle`. Este último foi lido para o programa GILD e gerado um conjunto de formulações ILP. Para além da formulação usando o modelo básico, que é apresentada pelo nome da descrição teste, foram realizadas outras sete formulação que têm na terminação uma combinação das letras `b, r` e `m` cujo o significado é:

b - formulação realizada apenas com variáveis binárias.

r - formulação com optimização do número de registos.

m - formulação considerando operações mult-ciclo.

A coluna *Cpx* da tabela representa um coefeciente de complexidade da formulação ILP. Este valor é calculado utilizando a seguinte expressão:

$$Cpx = 1 \cdot binVars + 2 \cdot intVars + 4 \cdot realVars$$

onde $binVars$, $intVars$ e $realVars$ representam, respectivamente, o número de variáveis binárias, inteiras e reais existentes na formulação e cujo valor tem que ser determinado. Ou seja, não estão previamente determinados pelas restrições impostas na formulação. Note-se que este valor não representa uma medida absoluta da complexidade da formulação, pois esta depende em grande parte do número e tipo de restrições existentes, ou seja, do espaço de procura.

Para os programas `opbdp` e `01solo` só fazem sentido as formulações que consideram todas as variáveis binárias, as outras soluções não têm qualquer significado.

Benchmark Names	Cpx	Execution time (sec.)			
		Cplex	lp_solve	opbdp	01solo
diffeq	17	n/a	0.2	0.6	(0.1)?
diffeqb	21	n/a	0.2	0.6	(0.1)
diffeqbm	25	n/a	0.2	0.6	(0)
diffeqbr	101	n/a	0.3	0.6	(0.1)
diffeqbrm	105	n/a	0.2	0.7	(0.1)
diffeqm	21	n/a	0.1	0.5	(0.1)
diffeqr	29	n/a	0.1	0.6	(0.1)?
diffeqrm	33	n/a	0.1	0.6	(0)
ellipf	36	n/a	0.2	0.7	(0.1)
ellipfb	37	n/a	0.3	0.7	(0.1)
ellipfbm	37	n/a	0.3	0.6	(0.1)
ellipfbr	222	n/a	0.6	0.9	(0.1)
ellipfbrm	222	n/a	0.7	0.9	(0.1)
ellipfm	36	n/a	0.2	0.6	(0.1)
ellipfr	63	n/a	0.3	0.8	(0.1)
ellipfrm	63	n/a	0.3	0.8	(0.1)
gcd	6	n/a	0.1	0.4	(0)?
gcdb	9	n/a	0.1	0.5	(0.1)
gcdbm	12	n/a	0.1	0.5	(0.1)
gcdbr	75	n/a	0.2	0.5	(0.1)
gcdbrm	78	n/a	0.1	0.6	(0.1)
gcdm	9	n/a	0.1	0.5	(0)?
gcdr	16	n/a	0	0.6	(0)?
gcdrm	19	n/a	0.1	0.6	(0.1)?
kalman	13	n/a	0.2	0.6	(0.1)?
kalmanb	17	n/a	0.2	0.7	(0.1)
kalmanbm	24	n/a	0.3	0.7	(0.1)
kalmanbr	223	n/a	0.4	0.9	(0.1)
kalmanbrm	230	n/a	0.5	0.9	(0.1)
kalmanm	20	n/a	0.1	0.7	(0.1)
kalmanr	43	n/a	0.1	0.7	(0.1)?
kalmanrm	50	n/a	0.2	0.7	(0.1)
tlc	2	n/a	0	0.5	(0.1)?
tlcb	3	n/a	0	0.5	(0.1)
tlcbm	3	n/a	0.1	0.5	(0.1)
tlcbr	48	n/a	0	0.5	(0.1)
tlcbrm	48	n/a	0.1	0.5	(0.1)
tlcm	2	n/a	0	0.4	(0)?
tlcr	9	n/a	0.1	0.5	(0)?
tlcrm	9	n/a	0.1	0.5	(0.1)?

n/a - resolução não disponível

() - erros durante a resolução

? - solução não encontradal

Tabela 3: Resultados em alguns circuitos de teste

5 Conclusões

Neste relatório descreveram-se três modelos de síntese de alto nível utilizando uma formulação ILP. O modelo básico permite apenas optimizar os sub-problemas de planeamento temporal e alocação de recursos. O modelo multi-ciclo realiza a mesma optimização mas considera que existem unidades funcionais que realizam as suas operações em mais que um passo de controlo. O terceiro modelo permite optimizar também o número de registos necessários para o circuito.

Foi ainda apresentado um ambiente para a síntese de alto nível de circuitos descritos em VHDL. Para isso foi necessário desenvolver uma ferramenta (GILD), para manipulação de grafos e geração de formulações ILP utilizando os modelos anteriormente descritos. Este programa representa internamente os grafos da descrição do circuito de forma a permitir facilmente a introdução de novos algoritmos. Quer para manipulação dos próprios grafos (DFG e/ou CFG), quer para a geração de modelos ILP mais complexos. Por exemplo, modelos que optimizem o número de buses e multiplexers de circuito, ou que considerem também os outros problemas em que usualmente de subdivide a síntese de alto nível.

A comparação da complexidade, em termos de variáveis, para os diferentes modelos de síntese de alto nível foi apresentada para um conjunto de circuitos de teste. A solução de cada um deste problemas foi efectuada por programas que resolvem problemas lineares genéricos (`Cplex` e `lp_solve`) ou que se restringem a variáveis binárias (`opbdp` e `01solo`). Devido à simplicidade quer dos modelos utilizados quer dos circuitos de teste envolvidos, não é possível fazer uma comparação sobre o melhor modelo/programa a utilizar. No entanto, espera-se que o programa desenvolvido no INESC/IST (`01solo`) incorpore técnicas de procura específicas que permitam tempos menores que os outros programas.

Referências

[1] M. Rim, R. Jain, and R. D. Leone, "Optimal allocation and binding in high-level synthesis," in *Proceedings of Design Automation Conference (DAC)*, pp. 120–123, 1992.

[2] S. Chaudhuri, R. A. Walker, and J. E. Mitchell, "Analyzing and exploiting the structure of the constraints in the ilp approach to the scheduling problem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, pp. 456–471, December 1994.

[3] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 10, pp. 464–475, April 1991.

[4] G. D. Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[5] J. Wilberg, A. Kuth, R. Camposano, W. Rosenstiel, and T. Vierhaus, "Design exploration in castle," in *Workshop on High Level Synthesis Algorithms Tools and Design (HILES)* (GMD-Studien, ed.), vol. 276, Stanvord University, November 1995.

[6] M. Berkelaar, *Unix Manual page of lp_solve*. Eindhoven University of Technology, Design Automation Section, michel@es.ele.tue.nl, 1992.

[7] CPLEX Optimization, Inc., *Using the CPLEX Basse System*. Version 4.0.

[8] P. Barth, "A davis-putnam based enumeration algorithm for linear pseudo-boolean optimization," tech. rep., Max-Planck-Institut Fűr Informatik, January 1995.

[9] L. J. Hafer and A. C. Parker, "A formal method for the specification, analysis and design of register-transfer level digital logic," *IEEE Transactions on Computer-Aided Design*, vol. 2, pp. 4–18, January 1983.

[10] B. Landwehr, P. Marwedel, and R. Dőmer, "Oscar: Optimum simultaneous scheduling, allocation and resource binding based on integer programming," in *Euro-DAC with Euro-VHDL*, 1994.