C 言語からの動作合成による回路設計と設計空間の探索

1150059 小松 達也（密山研究室）

1 はじめに

C 言語ベースの設計手法は、上位設計による設計効率の向上だけでなく、ハードウェア技術者にとっても使いやすく、必要に応じてハードウェアの最適化が行えなければならない。動作合成ツールによる生成回路は、C ソースの記述方法やツールのオプション設定などによって制御できることが求められる。本研究では、動作合成ツールとして Vivado HLS を用い、C 言語による動作合成の制御および合成条件指定によって得られる設計空間について評価を行った。

2 動作合成技術の概要

動作合成は動作レベルのアルゴリズム記述からレジスタ転送レベル (RTL) 記述を自動的に生成する技術であり、現在多くの動作合成ツールが開発されている [1]。動作レベル記述にさまざまな合成条件を与えることで、性能や回路規模についての最適化を行った RTL を自動生成することが可能である。

図 1 に動作レベル記述から合成されたデータフローグラフ (DFG) を示す。(a) の C ソースを加算器 1 個または加算器 2 個の制約下で動作合成した結果がそれぞれ (b) と (c) である。

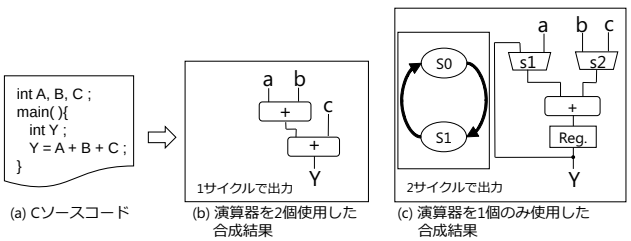

図 1: C 記述と演算資源制約に対応して合成された DFG

3 C 言語による動作合成の制御

設計者が意図するハードウェア実装を可能にするため、C 言語記述の書き換えによるメモリ (RAM) の構造操作や、ループ展開による並列化について実験的評価を行った。評価用アプリケーションとして、要素数は異なるが一次元配列同士の加算を行う C ソースを用いた。それぞれの動作合成結果を表 1 に示す。

まず、RAM の分割実装では、使用リソースが増加する一方でレイテンシが減少した。これは、RAM が増えたことでポートの総数が増加したためである。このことから、RAM の分割によって使用リソースおよびレイテンシの制御が可能であることがわかった。

次に、RAM の連結実装では、レイテンシが増加する一方で使用リソースが減少した。これは、LUT を使用して実装されていた分散 RAM を連結し、代わりに FPGA の内蔵メモリである BRAM を使用したためである。このことから、メモリ連結による BRAM 使用および使用リソースの制御が可能であることがわかった。

ループ展開による並列化では、逐次処理によるリソース共有ができなくなり使用リソースは増加したが、レイテンシは小さくなった。このことから、ループ展開の仕方によって使用リソースおよびレイテンシの制御が可能であることがわかった。

表 1: 動作合成結果比較

	使用リソース				レイテンシ
	BRAM_18K	DSP	FF	LUT	[cycle]
RAM 分割前	0	0	51	52	4
RAM 分割後	0	0	58	62	2
RAM 連結前	0	0	366	293	25
RAM 連結後	1	0	175	142	33
ループ未展開	0	0	59	14	9
ループ部分展開	0	0	58	45	5
ループ完全展開	0	0	50	48	2

4 合成条件による設計空間の探索

ソーベルフィルタ、DCT(IDCT)、FFT(IFFT)、AES の 4 つの評価アプリケーションを用い、合成条件を指定した場合の動作合成結果について、その演算リソースとレイテンシを評価した。動作合成結果を表 2 に示す。

表 2: 動作合成結果比較

	合成条件	使用リソース				レイテンシ
		演算器	DSP	FF	LUT	Latency[cycle]
Sobel	指定なし	14	32	5,309	6,665	3,096,073(100%)
	演算リソース最小	3	26	4,457	5,490	3,381,841(109%)
	ループのパイプライン化	15	29	4,501	5,503	2,349,649(76%)
	多重ループのパイプライン化	21	33	4,677	5,625	1,746,361(56%)
IDCT	指定なし	16	14	1481	1,677	544,809,089(100%)
	演算リソース最小	3	14	1,458	1,686	549,036,161(101%)
	ループのパイプライン化	20	14	1,760	1,781	112,197,633(21%)
	多重ループのパイプライン化	25	25	4,014	2,850	83,886,122(15%)
FFT	指定なし	55	67	8,046	14,816	97,359,892(100%)
	演算リソース最小	10	67	8,031	14,766	98,407,454(101%)
AES	指定なし	0	0	4,657	12,233	77,927(100%)
	ループのパイプライン化	0	0	4,246	12,233	76,972(99%)
	関数のパイプライン化	0	0	9,273	23,731	32,553(42%)

使用演算リソースを最小限に抑えた場合、どの評価アプリケーションにおいても演算リソースを削減できることを確認した。しかし、数値計算の処理内容が複雑になると演算部分で多くの IP ライブラリが用いられるため、IDCT および FFT ではレイテンシの変化はあまり見られなかった。

ループ内および関数内でパイプライン化を行った場合、演算リソースが増える一方でレイテンシが削減されるという結果が得られた。パイプライン化する処理が多くなるほどレイテンシの削減率が大きくなることを確認した。また、パイプライン化したい処理にメモリアクセスの競合やデータ依存性があるとパイプライン化できないことが確認できた。

5 まとめ

動作合成を用いた回路設計において、C ソース記述方法や、動作合成条件の指定によって得られる演算リソースとレイテンシのトレード･オフを評価した。今後は、Verilog-HDL を用いた RTL 設計と、動作合成ツールを用いた C ベース設計による実装結果を比較することにより、動作合成の効果的なコーディング手法についての検討を行う。

参考文献

[1] 三好健文:“ FPGA 向けの高位合成言語と処理系の研究動向”， コンピュータソフトウェア, Vol.30, No.1, pp.76-84, 2013 年 2 月.