Polynomial Methods for Allocating Complex Components

James Smith
Stanford University
Computer Systems Laboratory
Stanford, CA 94305

Giovanni De Micheli
Stanford University
Computer Systems Laboratory
Stanford, CA 94305

Abstract

Methods for performing component matching by expressing an arithmetic specification and a bit-level description of an implementation as word-level polynomials have been demonstrated for combinational circuits. This representation allows the functionality of a specification and existing implementation to be compared. We present extensions to this basic method that allow polynomial models to be constructed for circuits that employ sequential elements as well as feedback. Furthermore, we derive a means of approximating the functionality of non polynomial functions and determining a bound on the error of this approximation. These methods are used to synthesize an Infinite Impulse Response filter from a library of potential implementations.

1. Introduction

Comparing the functionality of a specification and implementation is a critical step in synthesis with complex components and design verification. This comparison often must be performed between arithmetic and bit-level abstractions of the functionality. Polynomial methods provide a means for generating word-level polynomial representations, given bit-level descriptions of an implementation. In generating a mathematical structure common to both levels of abstraction, allocation and verification of complex components can be performed, closing the semantic gap between specifications, such as those generated in MATLAB, and implementations, such as those modeled with Hardware Design Languages (HDLs).

Polynomial methods ([SmDe98]) have been proposed as an efficient technique for representing both arithmetic specifications and bit-level descriptions of existing implementations. These techniques have been applied to combinational circuits in synthesizing a JPEG encode block from existing subblocks. This paper extends those methods, presenting a mechanism for constructing word-level polynomial models for circuits that employ sequential elements, given a bit-level description of the circuit (Section 3). In addition, we present an algorithm for constructing an arithmetic model for those circuits that employ feedback (Section 4). The second part of the paper focuses approximating the functionality of circuits that implement non polynomial functions and bounding the error of this approximation (Section 5).

To illustrate the application of the algorithms developed in this paper, we compare the specification of a filter suitable for controlling the velocity of a tape through a tape drive to an existing filter, using polynomial methods (Section 6). The arithmetic specification for this filter is derived from MATLAB, while the existing implementation is described by Boolean equations or HDLs.

2. Related Work

Reusable blocks have traditionally been characterized imprecisely by verbal descriptions, such as “ethernet core” or “rasterizer”, and waveforms. Precise descriptions are usually restricted to small blocks such as adders and multipliers or combinational logic gates. Structures such as BDDs ([Br86]) provide a compact, canonical structure for representing such blocks. Binary Moment Diagrams ([BrCh95], [ChBr96]), Hybrid Decision Diagrams ([ClFu95]), and Multi Terminal BDDs ([ClFu93]) have provided extensions for dealing with more complex blocks, like adders and multipliers, but, like BDDs, can be exponentially large for blocks that implement non-linear functions. PHDDs ([ChBr97]) are well suited for representing the non linearities associated with floating point arithmetic, but are not readily extensible to other domains. [Mi96] introduced a method for modeling and manipulating circuits that implement polynomial functions using Zero-suppressed BDDs. This structure provides an efficient representation for those circuits for which a polynomial description is specified, but becomes exponentially large if discontinuities exist in the function.

The methods presented in [SmDe98] provided a mechanism for deriving a word-level polynomial representation for a complex design given a Boolean circuit description. This was achieved by determining the *minimum order* of a polynomial that was required to implement the same functionality as the circuit. Once this order was determined, coefficients were computed that caused polynomial output values in the integer domain to match circuit output values in the Boolean domain. These representations were shown to exist for all combinational blocks and were proven to be canonical with respect to circuit functionality. Furthermore, these algorithms were shown to be of polynomial complexity with respect to input word length.

The application domain of the methods presented in [SmDe98] excluded designs that contained synchronous elements or feedback paths. This paper will extend the application domain of polynomial methods by presenting algorithms for determining the *order* of the polynomial representation for synchronous circuits. As in [SmDe98], polynomial coefficients are computed by fitting the minimum order polynomial to the (x, y) coordinates derived from circuit input/output pairs. For example, a circuit with 4 bit input word x and 4 bit output word y, that is of order 2 and generates the input/output pairs {(0000, 0000), (0001, 0001), (0010, 0100)}, can be represented by the second order polynomial with coefficients [1, 0, 0]: $y = 1x^2 + 0x + 0 = x^2$. Coefficient computation is described in [SmDe98] and not revisited in this paper.

Algorithms for reducing the size of circuit representations by approximating circuit functionality have focused on minimizing the size of the representation and the number of input assignments for which the approximation and the actual circuit differ [RaMc98]. In generating approximations with word-level polynomial representations, our work focuses on minimizing polynomial size and the numerical distance between the implementation and the actual circuit.

3. Sequential Acyclic Circuits

In [SmDe98], it was established that polynomial representations, y = F(x), exist for all combinational circuits. This is due to the fact that combinational circuits specify a finite number of input/output pairs (x, y) that can be treated as coordinates to which a polynomial can be fit. Sequential circuits pose an additional problem because circuit outputs are not only a function of the current inputs but also previous inputs. Thus, a polynomial representation of a sequential circuit must contain terms that are dependent on previous input values: y = F(x, x@1, x@2, ..., x@m). The terminology x@*i* indicates a version of x that is delayed by *i* cycles.

3.1 Determining Combinational Equivalents

Generating a polynomial representation for sequential acyclic circuits can be achieved by determining an equivalent combinational circuit. A sequential circuit F(x), with an input $x \in B^n$, can be represented by a set of properly composed intermediate, combinational functions $f_i(x, f_0, f_1, ..., f_{i-1})$ between which registers (denoted by REG()) may exist. For example, the sequential circuit described by the following Verilog code, and shown in Figure 1, with input x and output y:

```
always @ (posedge clk) begin
        x_q <= x;
        y <= x + x_q;
end
```

can be similarly described by the following functions (where y = F(x)):

$f_0(x) = x;$
$f_1(x, f_0) = x + REG(f_0(x));$
$F(x) = f_1(x, f_0).$

The existence of a register on the output of an intermediate function $f_i(x, f_0, f_1, ..., f_{i-1})$ causes the inputs of $REG(f_i)$ to be independent of the inputs of the unregistered version of the intermediate function f_i. That is, the values of $\{x, f_0, f_1, ..., f_{i-1}\}$ are independent of one another on successive cycles. As a result, if *m* intermediate functions have registered outputs, the sequential function F(x) has an equivalent combinational function F(x, x@1, x@2, ..., x@m), where x@j, for $0 \le j \le m$, is the value of x after j cycles. Note that m is finite due to the restriction that the circuit does not have feedback. The equivalent combinational function can be determined by recursively creating a set of additional intermediate functions $f_{ij}(x@1, f_{0j}, f_{1j}, ..., f_{i-1j})$ for each intermediate function with a registered output. $REG(f_i(x, f_0, f_1, ..., f_{i-1}))$ is then replaced by $f_{ij}(x@1, f_{0j}, f_{1j}, ..., f_{(i-1)j})$, $f_{i-1}(x, f_0, f_1, ..., f_{i-2})$ by $f_{(i-1)j}(x@1, f_{0j}, f_{1j}, ..., f_{(i-2)j})$, etc., in each intermediate function. In the example above, this would result in the following set of intermediate functions that described the combinational implementation of F(x):

$f_0(x) = x;$
$f_{01}(x@1) = x@1;$
$f_1(x, f_0) = x + f_{01}(x@1);$
$F(x) = f_1(x, f_0).$

A diagram of this transformation is shown in Figure 1.

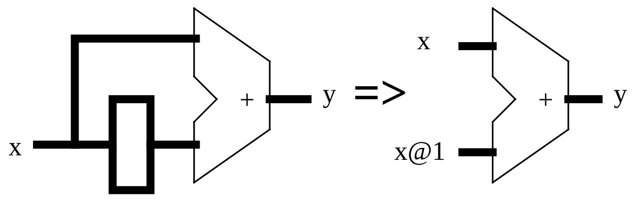


Fig. 1 Transformation of a sequential adder into a combinational circuit.

The output of intermediate functions f_i for which additional intermediate functions f_{ij} are created may go unused in the equivalent combinational circuit. For example, $f_0(x)$ is unused in the above representation of F(x, x@1) and can be pruned from the set of intermediate functions. Pruning is performed after all registers have been removed from intermediate equations. Note that the existence of the registered intermediate function $REG(f_i(x, f_0, f_1, ..., f_{i-1}))$ will not cause $f_i(x, f_0, f_1, ..., f_{i-1})$ to be pruned from F(x, x@1, x@2, ..., x@m) if an unregistered version of $f(x, f_0, f_1, ..., f_{i-1})$ is used elsewhere.

A polynomial representation for F(x) can now be determined from F(x, x@1, x@2, ..., x@m). The order of F(x, x@1, x@2, ..., x@m) is determined with respect to each x@j, for $0 \le j \le m$, as independent variables, and the coefficients of the polynomial representation are determined. In the previous example, this would result in F(x) = x + x@1.

4. Sequential Cyclic Circuits

The method for determining polynomial representations for sequential acyclic circuits relied on the acyclic nature of the circuit to guarantee that a finite number of time-shifted inputs were required. However, by breaking the feedback path of a cyclic circuit F(x) , the previous techniques can be used to derive the order of the cyclic circuit. This is achieved by introducing an input $F_{feedback}(x)$, and determining the order of $F(x, F_{feedback})$ with respect to x and $F_{feedback}$.

4.1 Order Computation with Feedback

Assume a function F(x) contains two branches, one branch with feedback and one initialization branch without feedback (Figure 2):

initial $F(x) = f_1(x);$
always $F(x) = f_2(x, F_{feedback}(x));$

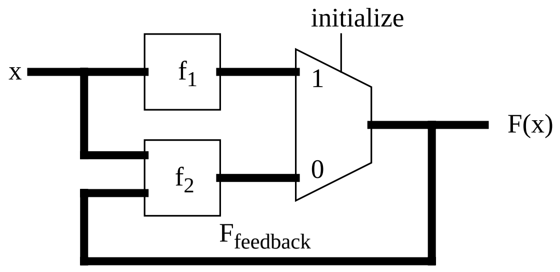


Fig. 2 Sequential cyclic circuit model

The order of $f_1(x)$ with respect to x, referred to as n_1, can be determined using the techniques presented in Section 3. Furthermore, if $y = F_{feedback}(x)$ is treated as an input within the second branch, then the order of $f_2(x, y)$ with respect to x, referred to as n_2, and with respect to y, referred to as m, can also be determined. After initialization, the order of F(x) is n_1, and after the first iteration of the feedback branch, the order of F(x) is less than $(mn_1 + n_2)$ and greater than mn_1. In general, the order of F(x) is less than $m \cdot O(F_{feedback}(x)) + n_2$ and greater than $m \cdot O(F_{feedback}(x))$. Thus, the upper bound on the order after k iterations of the feedback branch, is:

$$m^k \cdot n_1 + \sum_{i=0}^{k-1} m^i \cdot n_2$$

To determine the order of F(x) there are three cases that need to be considered:

(1) k is a constant,

(2) k is not a constant, m = 1, and there is no $x \cdot y$ term in the polynomial $f_2(x, y)$,

(3) k is not a constant, and ($m \neq 1$ or there is no $x \cdot y$ term in the polynomial $f_2(x, y)$).

In *case (1)*, the order of F(x) can be bounded according to the equation above. In *case (2)*, the order of F(x) is the greater of n_1 and n_2. For both of these cases, since the order of F(x) is independent of k, a polynomial representation exists for F(x). In *case (3)*, the order of F(x) is dependent on k and is therefore unbounded.

To analyze case (3), assume that the specification S(x) is modeled similar to Figure 2, as:

initial $S(x) = s_1(x)$;
always $S(x) = s_2(x, S_{feedback}(x))$;

If S(x) has bounded order n_s then the equation:

$$m^k \cdot n_1 \le n_s \le m^k \cdot n_1 + \sum_{i=0}^{k-1} m^i \cdot n_2$$

can be solved for k, using numerical methods, to determine bounds on k within which F(x) can have the same order as S(x), and therefore possibly implement S(x). If S(x) has unbounded order, then S(x) is implemented by F(x) if and only if $s_1(x) = f_1(x)$ and $s_2(x, S_{feedback}(x)) = f_2(x, F_{feedback}(x))$. Thus, even if a function F(x) does not have a bounded order, and therefore no polynomial representation, it can still be compared to a specification S(x) by comparing the initialization and feedback polynomials of S(x) and F(x).

Example 4.1.1 Consider a Boolean circuit F(x, y) with inputs x, y, output z, and a base case polynomial and recursive polynomial as follows:

initial begin	always begin
z = x;	z = z + x;
d = y;	d = d - 1;
end	end

Breaking the feedback loops introduces variables $z_{feeback}$ and $d_{feedback}$ and results in computation of the following set of polynomials:

initialize = 1	initialize = 0
z = x	$z = z_{feedback} + x$
d = y	$d = d_{feedback} - 1$

Since the feedback polynomial is of order one with respect to $z_{feedback}$ (m = 1) and contains no $xz_{feedback}$ term, case 2 is satisfied, and the order of F(x, y) with respect to x is the greater of n_1 and n_2, both of which are 1.

5. Approximations

Polynomial representations are an efficient way to encapsulate the functionality of arithmetic circuits. Circuits that do not implement polynomial functions can be modeled by determining subdomains over which the circuit does implement polynomial functionality, as outlined in [SmDe98]. However, this becomes very expensive when the number of subdomains becomes large. Circuits that approximate continuous functions frequently generate many subdomains. For example, a circuit that implements $F(x) = \lfloor x/2 \rfloor$, where x is an n bit word, requires 2^{n-1} subdomains (Figure 3). Rather than represent F(x) as a list of subdomains of x and corresponding polynomials that describe F(x) exactly over those subdomains, it is much more efficient to represent F(x) as the polynomial x/2 and specify the maximum error between the continuous function x/2 and F(x).

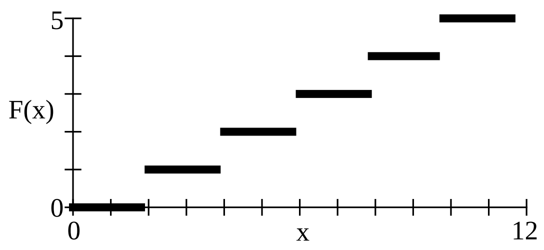
5.1 Computing Approximations

As proven in [SmDe98], the order of a function is reduced by one by computing the difference F(x+1) - F(x). If the order of F(x) is n, then recursively performing this difference n+1 times will reduce the function to 0. For example, consider F(x) = x:

Step 1: F(x+1) - F(x) = (x+1) - x = 1
Step 2: (F(x+2) - F(x+1)) - (F(x+1) - F(x)) = 1 - 1 = 0.

However, if performing this difference n times results in a function that is not zero, but has a small range of output values, then the function can be approximated well by a polynomial of degree n.

The above fact can be translated to approximating a set of Boolean equations with a polynomial. If an output bit vector y is m bits long and the upper k bits of y are 1 then $y > -2^{m-k}$. Similarly, if the upper k bits of $y - 2^{m-k}$ (performed using two complement arithmetic) are 1, then $y < 2^{m-k}$. As, a result, if F^- is defined to be F(x+1) - F(x) and F^+ is defined



Domain	Polynomial
[0, 1]	F(x) = 0
[2, 3]	F(x) = 1
[4, 5]	F(x) = 2
[6, 7]	F(x) = 3
[8, 9]	F(x) = 4
...	...

Fig. 3 Subdomains generated by the function F(x) = x/2.

to be F(x+1) - F(x) - 2^{m-k}, then the following statement holds: if the upper k bits of (F^- or F^+) are 1, then $-2^{m-k} < F(x+1) - F(x) < 2^{m-k}$. The bound on F(x+1) - F(x) (termed G(x)), allows us to derive an approximation of F(x):

Given F(x+1) - F(x) = G(x)
Given F(0)
=> F(1) = G(0) + F(0)
=> F(2) = G(1) + F(1) = G(1) + G(0) + F(0)
...

$$=> F(x) = \left(\sum_{i=0}^{x-1} G(i)\right) + F(0)$$

Assuming G(i) is small, F(x) can be approximated by:

$$F_{approx}(x) \approx x \cdot \left(\left(F(2^n - 1) - F(0)\right)/2^n\right) + F(0)$$

5.2 Computing Approximation Error

The difference between F(x) and $F_{approx}(x)$, termed Δ(x), is:

$$\Delta(x) = \sum_{i=0}^{x-1}\left(G(i) - \left(F(2^n - 1) - F(0)\right)/2^n\right)$$

Since $-2^{m-k} < G(i) < 2^{m-k}$, it requires only m-k bits to represent G(i). Assuming $m \approx k$ for a good approximation, computation of $G(i) \cdot 2^n - F(2^n - 1) - F(0)$ need only be performed only over a short word length (m-k bits). Defining δ(i) = $G(i) \cdot 2^n - F(2^n - 1) - F(0)$ yields:

$$\Delta(x) \cdot 2^n = \sum_{i=0}^{x-1}\left(G(i) \cdot 2^n - F(2^n - 1) - F(0)\right) = \sum_{i=0}^{x-1} \delta(i)$$

Expressing δ(i) as a sum of its bits $\delta_j(i)$ yields:

$$\Delta(x) = \sum_{i=0}^{x-1}\left(\sum_{j=0}^{n+m-k-1} 2^{j-n} \cdot \delta_j(i)\right)$$

A positive bound on Δ(x) can then be determined from each bit $\delta^+_j(i)$ of the positive values of δ(i):

$$\Delta^+(x) < \sum_{i=0}^{x-1}\left(\sum_{j=0}^{n+m-k-1} 2^{j-n} \cdot \left(\delta^+{}_j(i) \neq 0\right)\right)$$

Similarly, a negative bound can be determined from each bit $\delta^-_j(i)$ of the negative values of δ(i):

$$\Delta^-(x) > \sum_{i=0}^{x-1}\left(\sum_{j=0}^{n+m-k-1} 2^{j-n} \cdot \left(\delta^-{}_j(i) \neq 1\right)\right)$$

$\delta^+_j(i)$ and $\delta^-_j(i)$ are determined by computing the positive and negative cofactor of $\delta_j(i)$ with respect to $\delta_{n+m-k}(i)$ (because if $\delta_{n+m-k}(i)$ is zero, δ(i) is positive and if it is one, δ(i) is negative).

Computing Δ(x) for all 2^n values of x is prohibitively complex due to the size of the domain and the fact that Δ(x) is a summation of x values. To circumvent this summation and determine a bound on Δ(x), the maximum values for the following are determined:

Δ(x+1) - Δ(x) where bit $x_0 = 0$
Δ(x+2) - Δ(x) where bits x_0, $x_1 = 0$
...
$\Delta(x+2^{n-1}) - \Delta(x)$ where bits $x_0, x_1, ..., x_{n-1} = 0$

The sum of the maximum values of each of the above equations provides a bound on the error of the approximation, since any value of Δ(x) can be reached by summing a subset of the above equations. For example:

Δ(7) = [Δ(6+1) - Δ(6)] + [Δ(4+2) - Δ(4)] + [Δ(0+4) - Δ(0)].

If a suitable bound is found for the nth computation of F(x+1) - F(x), termed $^n(x)$, rather than the first computation of F(x+1), then an approximation for F(x) can be computed within Δ from Newton's forward difference interpolating formula:

$$F(x) = F(0) + \binom{x}{1}\hat{F}(0) + \binom{x}{2}\hat{F}^2(0) + ...$$

$$+ \binom{x}{n}\hat{F}^n(0) + \binom{x}{n+1}\frac{\hat{F}^n(2^n - 1) - \hat{F}^n(0)}{2^n}$$

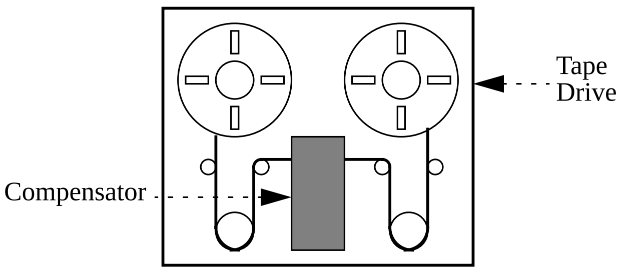
Example 5.2.1 Consider the eight-bit function y = F(x) where $x \in B^8$ and $y \in B^8$:

$y_0 = x_1$;	$y_4 = x_5$;
$y_1 = x_2$;	$y_5 = x_6$;
$y_2 = x_3$;	$y_6 = x_7$;
$y_3 = x_4$;	$y_7 = 0$;

This circuit could be partitioned into 64 subdomains and represented exactly with 64 order 0 polynomials (similar to Fig. 3). However, the first order iteration yields the bound -1 < F(x+1) - F(x) < 1. Therefore, F(x) can be approximated by the first order polynomial $F_{approx}(x) = x(F(2^8-1) - F(0))/2^8 = .498x$. The resulting $(\Delta(x+1)-\Delta(x))2^8$ is:

$\delta_0 = 1$;	$\delta_5 = 0$;
$\delta_1 = 0$;	$\delta_6 = 0$;
$\delta_2 = 0$;	$\delta_7 = 1$;
$\delta_3 = 0$;	$\delta_8 = x_0$';
$\delta_4 = 0$;	$\delta_9 = x_0$';

Evaluating at $x_0 = 0$, the negative error contributed by Δ(x+1)-Δ(x) is $-127(2^{-8})$ = -.5 units. The positive error contributed is zero units since $(\Delta(x+1)-\Delta(x))2^8 = -127$ when $x_0 = 0$. Other differences $\Delta(x+2^i)-\Delta(x)$ contribute



Transfer Function:

$$H(z) = \frac{.094 - .28z^{-1} + .19z^{-2} + .19z^{-3} - .28z^{-4} + .094\ z^{-5}}{1 - 5z^{-1} + 10z^{-2} - 10z^{-3} + 5z^{-4} - z^{-5}}$$

Fig. 4 Digital filter used as a compensator for controlling the move of a tape through a tape drive

no negative error and a total of .5 units of positive error, resulting in the error bound: $-.5 < \Delta < .5$. Thus, the circuit implements the polynomial $F(x) = .498x$ within .5 units and could be used to match the specification $S(x) = x/2$. This approximate representation is far less complex than the 64 polynomials that would be required to represent the circuit exactly.

6. Application

Many embedded applications require digital filters to control mechanical operations. Common examples include altitude control systems for satellites, yaw dampers in airplanes, and fuel injection controllers in automobiles. We will apply polynomial methods to determine an existing filter, from a library of filters, suitable for reuse in a tape drive controller (Figure 4). The velocity of the tape with the tape drive is controlled by a voltage applied to the reel motor. This voltage is a function of past velocities, and therefore past voltages, as well as the displacement required to position the tape properly. An existing circuit implementation within the library of filters is shown in Figure 5, with combinational blocks already described by polynomials. The challenge is to determine if the circuit can be allocated to implement the following specification, generated from MATLAB:

$$\begin{aligned} S(x) = {} & 5S(x@1) - 10S(x@2) + \\ & 10S(x@3) - 5S(x@4) + S(x@5) + \\ & .09375x - .28125(x@1) + .1875(x@2) + \\ & .1875(x@3) - .28125(x@4) + .09375(x@5) \end{aligned}$$

The first step in generating a polynomial representation for the circuit described in Figure 5 is to break the feedback paths. This results in $F_{feedback}$ replacing F in the list of equations and being added to the list of inputs. The next step in generating a polynomial representation requires removal of all registers. After one iteration of register removal, x@1 replaces REG(x) and is added to the input list. Subsequently, x_q@1 replaces REG(x_q) and the equation x_q@1 = x@2 is added to the list of equations, and x@2 is added to the input list. Register removal results in a circuit with the following inputs: {x, x@1, ..., x@5, $F_{feedback}$, $F_{feedback}$@1, ..., $F_{feedback}$@5}. The equations for each of {x_q, ..., x_qqqqq, F_q, ..., F_qqqqq} are pruned from the list of equations as

```
input: x                      output: F
x_q = REG(x)                  F_q = REG(F)
x_qq = REG(x_q)               F_qq = REG(F_q)
x_qqq = REG(x_qq)             F_qqq = REG(F_qq)
x_qqqq = REG(x_qqq)           F_qqqq = REG(F_qqq)
x_qqqqq = REG(x_qqqq)         F_qqqqq = REG(F_qqqq)

H1 = 160F_q - 320F _qq + 320F_qqq
H2 = - 160F_qqqq + 32F_qqqqq
H3 = x - 3x_q + 2x_qq + 2x_qqq
H4 = 3x_qqqq + x_qqqqq
H = H1 + H2 + H3 + H4
F = H>>5
```

Fig. 5 Circuit description for library element to be compared to tape controller specification

those variables are no longer used to compute F (the have been replaced by {x@1, ..., x@5, $F_{feedback}$@1, ..., $F_{feedback}$@5}). The complete set of resulting equations is:

$$\begin{aligned} & H1 = 160F_{feedback}@1 - 320F_{feedback}@2 + 320F_{feedback}@3 \\ & H2 = -160F_{feedback}@4 + 32F_{feedback}@5 \\ & H3 = x - 3x@1 + 2x@2 + 2x@3 \\ & H4 = 3x@4 + x@5 \\ & H = H1 + H2 + H3 + H4 \\ & F = H>>5 \end{aligned}$$

At this point, the circuit description has no feedback paths and no registers.

Order computation with respect to each of {$F_{feedback}$, $F_{feedback}$@1, ..., $F_{feedback}$@5} results in an order of one for each input. However, the order of the circuit with respect to each of {x, x@1, ..., x@5} is very large, indicating that a representation of an approximation of this circuit will be more efficient. Computation of F(x+1, x@1, ...) - F(x, x@1, ...) reveals that -1 < F(x+1, x@1, ...) - F(x, x@1, ...) < 1. A similar result is determined for x@1, ..., x@5. Thus, the term that each of {x, x@1, ..., x@5} contributes to the polynomial representation of the circuit can be represented by an approximation of order 1, of the form $x(F(2^n-1) - F(0))/(2^n-1)$. Following the error quantification steps outlined in Section 5, the bound on the error contributed by approximating each term of the polynomial that contains one of {x, x@1, ..., x@5} is $-.968 < \Delta < .968$. After performing coefficient computation, the following polynomial representation for the circuit is determined:

$$\begin{aligned} F(x) = {} & 5F_{feedback}(x@1) - 10F_{feedback}(x@2) + \\ & 10F_{feedback}(x@3) - 5F_{feedback}(x@4) + \\ & F_{feedback}(x@5) + .093749x - .281246(x@1) + \\ & .18749(x@2) + .18749(x@3) - .281246(x@4) + \\ & .093749(x@5) \end{aligned}$$

After closing the loop by setting $F_{feedback} = F$, the specification S(x) and implementation F(x) can be compared by comparing their representative polynomials. The coefficients of S(x) and F(x) do not match exactly, due to the approximation of F(x), but are the same within 10^{-4}. Thus, the existing circuit can be allocated to implement the specification if the circuit tolerance of 10^{-4} is acceptable.

Accumulator Stages	Number of Registers	Exec. Time
1	16	7.76s
2	32	25.84
3	48	79.47
4	64	177.50
5	80	326.19

Fig. 6(a) Execution time required for register removal on 16 bit accumulators.

Word Sizes	Exec. Time
4	0.01s
8	0.05
16	0.19
32	1.11
64	9.14
128	76.80

Circuit Function	Approx. Error
x/8	0.87
x/4	0.75
x/2	0.50
3x/4	1.75
7x/8	1.87

Fig. 6(b) Execution time for determining an approximation to the function x/2 **(c)** Accuracy of approximation for several 16 bit functions.

7. Experimental Results

Experiments in [SmDe98] verified that polynomial methods, for combinational circuits, were of polynomial complexity with respect to input bit width. To quantify the performance of polynomial methods for synchronous circuits, experiments were conducted, on a 200MHz Indy with 256MB of memory, to gauge the relationship between the execution time required to generate equivalent combinational circuits and the number of registers (Figure 6(a)). The circuits on which this was performed were 16 bit accumulators with between one and 5 register stages (i.e. F(x) = x + x@1, F(x) = x + x@1 + x@2, ..., F(x) = x + x@1 + ... + x@5). Execution time varied quadratically with the number of registers. Note that the register removal tool is written in Perl and the execution times in Figure 6(a) can be reduced greatly using compiled code.

Further experiments were conducted to determine the execution time of circuit approximation relative to input bit width. Polynomial approximations were computed for the circuit that implements the function y = x/2 for input bit widths ranging from 4 to 128 bits (Figure 6(b)). While of high order complexity, approximations completed quickly, even for the widest datapaths. The accuracy of circuit approximation was determined for several circuits of bit width 16 (Figure 6(c)), all of which resulted in an error of less than 2 units over the range $[0, 2^{16}-1]$. These experiments were performed with compiled code.

8. Conclusion

In performing high level synthesis and reusing existing designs, automating allocation requires a means for quickly determining whether an existing block performs the function outlined in the specification. Current methods for completing this task become prohibitively memory intensive or time consuming for circuits that implement complex functions. Previous work demonstrated an algorithm for performing component matching efficiently by constructing polynomial representations for combinational circuits.

In this paper, we have developed extensions to polynomial methods that allow polynomial representations to be constructed for synchronous circuits, including those with feedback paths. Furthermore, we have developed an algorithm for determining an approximate polynomial representation for those circuits that contain many discontinuities and a means for quantifying the error of that approximation. Finally, we demonstrated an application of polynomial methods in which an existing filter design was matched to the specification of a tape drive controller.

Future work will focus on developing partitioning algorithms that allow polynomial representations to be computed for less complex circuit partitions. The ease of composition of polynomial representations will then allow the order of the overall circuit to be computed with reduced complexity with respect to input lengths.

Acknowledgments

This work is funded in part by MARCO and ARPA.

References

[SmDe98] J. Smith and G. De Micheli, “Polynomial Methods for Component Matching and Verification”, Proceedings of the ACM/IEEE International Conference on Computer Aided Design, 1998. Prior to ICCAD 98, this paper can be found on the web: http://aglaia.stanford.edu/SmDe98.html.

[Br86] R. Bryant “Graph Based Algorithms for Boolean Function Manipulation”, IEEE Transactions on Computers, C-35(8), 1986.

[BrCh95] R. Bryant and Y.A. Chen, “Verification of Arithmetic Circuits with Binary Moment Diagrams”, Proceedings of the 32nd ACM/IEEE Design Automation Conference, p. 535 - 541, 1995.

[ChBr96] Y.A. Chen and R. Bryant, “ACV: An Arithmetic Circuit Verifier”, Proceedings of the ACM/IEEE International Conference on Computer Aided Design, p. 361-365, 1996.

[ClFu95] E.M. Clarke, M. Fujita, and X. Zhao, “Hybrid Decision Diagrams”, Proceedings of the ACM/IEEE International Conference on Computer Aided Design, p. 159 - 163, 1995.

[ClFu93] E.M. Clarke, K. McMillan, X. Zhao, M. Fujita, and J. Yang, “Spectral transformations for Large Boolean Functions with Applications to Technology Mapping”, Proceedings of the 30th ACM/IEEE Design Automation Conference, IEEE Computer Society Press, 1993.

[ChBr96] Y.A. Chen and R. Bryant, “*PHDD: An Efficient Graph Representation for Floating Point Circuit Verification”, Proceedings of the ACM/IEEE International Conference on Computer Aided Design, p. 2-7, 1997.

[Mi96] S. Minato, “Implicit manipulation of Polynomials Using Zero-Suppressed BDDs”, 1996.

[RaMc98] K. Ravi, K. McMillan, T. Shiple, F. Somenzi, “Approximation and Decomposition of Binary Decision Diagrams”, Proceedings of the 35th ACM/IEEE Design Automation Conference, IEEE Computer Society Press, 1998.