CBD algebraic + discrete-time

Angela Mizero Jordan Parezys

November 7, 2018

1 Introduction

The task of this assignment was to implement some functions that a CBD block must implement. After implementing the functions, we needed to create a CBD simulator that contains

- At least one inner CBD
- At least one linear algebraic loop.
- At least one Delay block.

2 Implemented methods (Angela Mizero)

2.1 getDependencies

The getDependcies function returns all the blocks that a CBD block depends on. However the getDepencies function returns returns different values depending on the block type. For a delay block the getDepencies function differs. For the other blocks it returns all the blocks that have an input signal going to the block for which we want to determine dependencies. For a delay block, at the first iteration, it depends on the IC which is the initial value. At the other iterations, it does not depend on anything else.

2.2 compute

The compute function takes the inputs of a block, applies a computation on them, and appends the result to the output signal. The blocks types that were used in this assignment are:

1. DelayBlock This block returns its IC value at the first iteration and for the other iterations returns the input value of the previous iteration.
2. ConstantBlock This block returns its own input value as output.
3. InverterBlock This returns the inverse of its input.

4. AdderBlock This block adds its inputs.
5. ProductBlock This block multiplies its inputs.
6. RootBlock This block takes the input1-th root of input2.
7. ModuloBlock This block produces the result of a modulo operation between two inputs.
8. GenericBlock This block applies a given mathematical function to its input.

2.3 _createDepGraph

The _createDepGraph function includes an accept_cbd function which accepts a CBD and gets all the blocks from the CBD. After getting all the blocks from the CBD. It gets their dependencies by calling the getDependecies method and sets their dependencies in the DepGraph class. However some blocks have sub blocks that also need to be included in the construction of the graph. Therefore the accept_cbd function must be recursively called if a block contains sub blocks.

2.4 _isLinear

This function checks if the equation produced is linear. An equation is linear if all of the points for any one equation lie on the same line. Linear equations graph are straight lines. Therefore, the InverterBlock, RootBlock, ModuloBlock cannot produce linear equations. Also, no variable in a linear equation is raised to a power greater than 1. Therefore, all the other blocks except the ProductBlock produce linear equations. When it comes to the ProductBlock, it produces linear equations if and only if it receives inputs which contain at most one unknown. Therefore the ProductBlock, must have at most one input coming from the algebraic loop. It can have more than one input coming from outside the algebraic loop, because those inputs are considered known since they are calculated beforehand.

3 CBD simulator (Jordan Parezys)

Our CBD simulator simulates a CBD which combines a Fibonacci serie CBD and a counter CBD. To satisfy the requirements of the assignment as described in the Introduction, the simulator contains more than one inner CBD delay block and algebraic loop. The main CBD contains the following sub CBDs:

1. counter: produces a series of numbers. The next number is the previous number incremented by one.
2. fibonacci: Produces a fibonacci series like so: 1 1 2 3 5 8 13 .....

3. fibonacci_divider: Combines the counter and the fibonacci series CBDs into one large CBD.

All the CBDs are shown in the figures below.

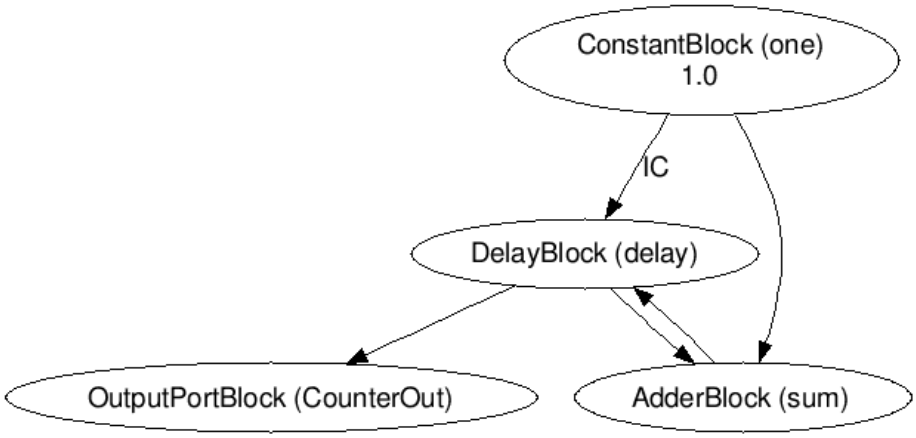

Figure 1: counter CBD

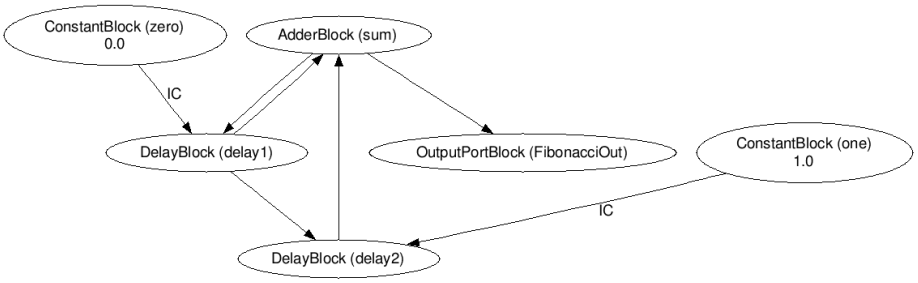

Figure 2: Fibonacci CBD

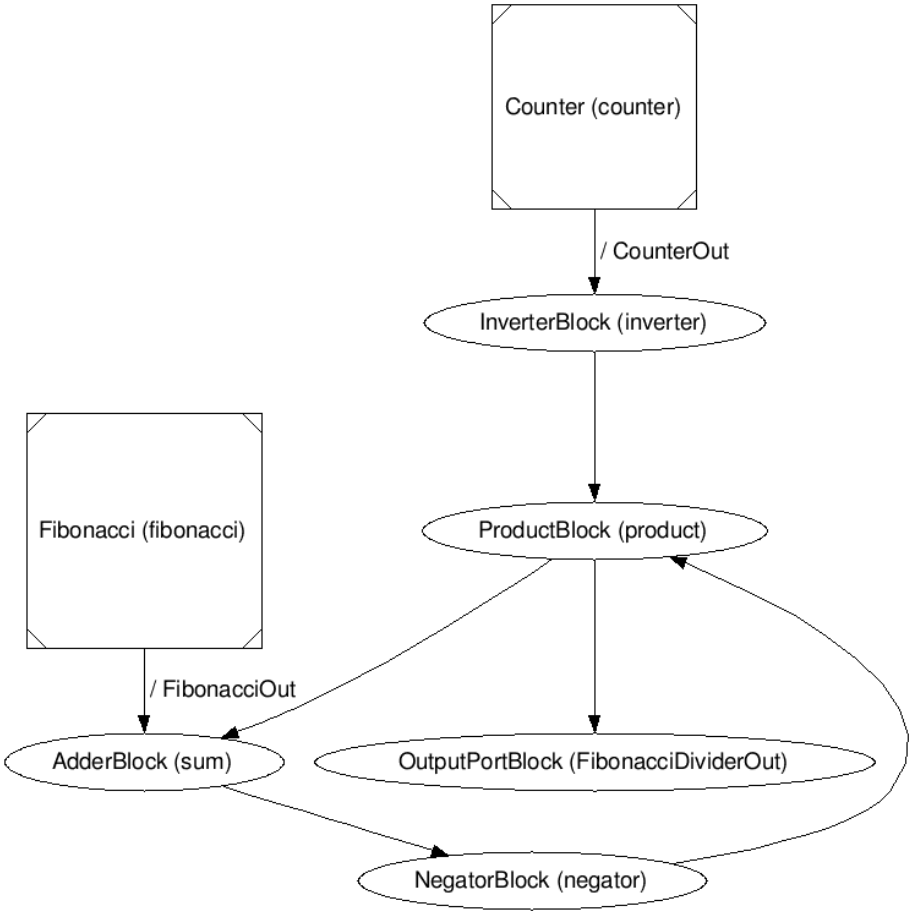

Figure 3: Divider CBD

The latex file that shows the equations of the various CBDs is included below.

4 Counter

$$\begin{cases} delay(0) = 1.0 \\ delay(i) = sum(i-1) \\ sum(i) = 1.0 + delay(i) \\ one(i) = 1.0 \end{cases}$$

5 Fibonacci

$$\begin{cases} delay1(0) = 0.0 \\ delay1(i) = sum(i-1) \\ delay2(0) = 1.0 \\ delay2(i) = delay1(i-1) \\ sum(i) = delay1(i) + delay2(i) \\ zero(i) = 0.0 \\ one(i) = 1.0 \end{cases}$$

6 FibonacciDivider

$$\begin{cases} sum(i) = product(i) + fibonacci(i) \\ negator(i) = -sum(i) \\ product(i) = negator(i) * inverter(i) \\ inverter(i) = \frac{1}{counter(i)} \end{cases}$$