Quadratic Polynomial Tutorial

SRC Deliverables Draft, Fall 2009

1. Introduction

In this tutorial, we will be creating a simple PyTDL design that computes a quadratic polynomial of the form:

$$result = x^2 + bx$$

Where we can arbitrarily choose b. We'll select $b = 20$.

The purpose of this is to demonstrate the basic syntax of a working design.

1.1. Take-Away

We hope that this tutorial shows a simple stream calculator. Its sole purpose is to show the structure of both PyTDL designs and the associated constraint files. We also hope that it will provide a simple testbed for getting your simulation environment set up.

2. The Design Stage

The requirements for the design are:

- Accept a single stream of input x
- Perform the listed calculations on it

2.1. The First Pass

We will design the stream such that there are two separate paths. It will square the input in parallel to multiplying by the constant. We do not share resources here, since these two operations can be realized using special squaring and multiply-by-constant circuits.

The code is listed below.

```
from Int import Int, GlobalIn, GlobalOut
class Quadratic:
  def __init__(self):
        self.x = GlobalIn(16)
        self.bx = Int(16)
        self.xsquared = Int(16)
        self.result = GlobalOut(16)

  def dispatch(self, trigger="tInput"):
        create(tXReady)

  def square(self, trigger="tXReady"):
        self.xsquared = self.x * self.x
        create(tXSquared)

  def bx(self, trigger="tXReady"):
        b = Int(16, 20)
        self.bx = b * self.x
        create(tBX)

  def sum(self, trigger="tXSquared and tBX"):
        self.result = self.bx + self.xsquared
        create(tOutput)
```

We see that the dispatch rule accepts *x* on the token `tInput` and sends it as token `tXReady`. Since there are two rules, `square` and `bx`, which both accept `tXReady`, a fork arbiter will be automatically inserted to duplicate the token.

Once the two calculations have been completed, we need to sum the two parts. We do so using the `sum` rule which waits for both tokens to arrive (with their associated payloads) before doing the addition. The operation dependency is built into the design, and since it is latency tolerant, we do not need to worry about timing – the Join/And arbiter will handle the synchronization for us.

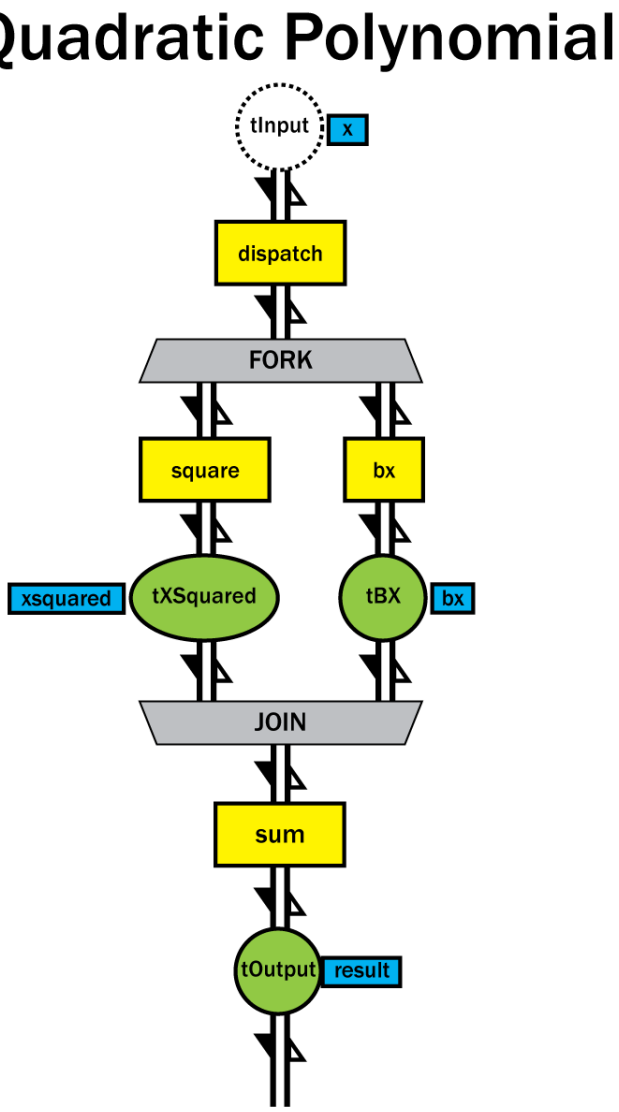
The constraints file is straightforward:

```
input = (tInput[x])
output = (tOutput[result])
```

We simply declare the input token associated with data `x` and the output token associated with the `result`.

Note that this is one of the rare situations where it is acceptable to break the requirement that input data have their own variable names. There are no conflicts or ambiguities in this design.

The design is visualized in the figure below.



2.1.1. Synthesize the Design

Synthesize the design with this command:

```
python PyTDL.py -o constraints.tc Quadratic.py
```

(See the User's Guide for details.)

You can now use the packaged test bench to run the simulation and verify correctness.