VisLang: A graphical programming language

Bryant Eisenbach

June 9, 2015

1 Introduction

VisLang is a block diagram language designed to allow fast and easy prototyping of programs for embedded processors. The language is created with a graphical editor in mind, and the core elements of the language are extensible so that any graphical editor can add additional elements for graphical display or other features.

2 Key Language Features

The language itself is based on the idea of blocks: small parts that can be grouped together into ever larger blocks and re-used across a program or programs. A small group of fundamental (or atomic) blocks will be defined and understood by the compiler for this language. Other blocks will be constructed as configurations of these atomic blocks. A standard library of useful functions will be constructed from these atomic blocks containing common parts such as timers, latches, etc.

Side effects in the produced code will be minimized by the combination of a strong type system and bounded code execution. The type system of VisLang supports common datatypes such as bool, single, double, signed and unsigned integers, as well as static arrays and structures of these simple datatypes. Bounded code execution is guarenteed through the restriction of for loops to a static size. This works well as embedded programs should only need to parse large buffers for digital busses, which are usually defined as being a static size. The language will contain methods for defining digital message structures and how to parse those structures into usable variables. Methods for parsing both packet-based (e.g. Ethernet) and word- based (e.g. RS232) will be provided by the standard library.

Lastly, time variance will be something provided fundamentally by the language. The time-step between subsequent iterations will be maintained in every program and provided to the user as a fundamental part. Most of the fundamental parts will be time-invariant, but this language feature will provide users with the ability to create dynamic parts that will care about time as a measured quantity.

3 Syntax

4 Example Program

Listing 1: ../example/timed-blinking-light/timed-blinking-light.vs

```
<Style Tag>
<Input scope=device name="digital_input_1" type=boolean>
    <!-- Hardware address 123 is DI_1 for ATMega328 -->
    <!-- TODO: Find real hardware address for this DI -->
    <Address>0x0123</Address>
</Input>
<Signal scope=global name="timer_reset" type=boolean/>
<Signal scope=local name="not_gate_output" type=boolean/>
<Signal scope=local name="or_gate_output" type=boolean/>
<Signal scope=local name="count_expired_lp" type=boolean/>
<NOT>
    <!-- All gates have input(s) and an output to connect to -->
    <Connection to="input" from="digital_input_1">
        <!-- A GUI Program could specify the shape of the connection here -->
        <!-- Not relevant for the compiler -->
    </Connection>
    <Connection to="not_gate_output" from="output"/>
</NOT>
<MEM initial_condition=0>
    <!-- Memory block would store the state each pass of the variable
         specified by current_pass_value at the end of execution
         such that the last_pass_value can be used in the local scope
         without suffering from algebraic loops -->
    <Connection to="current_pass_value" from="count_expired"/>
    <Connection to="count_expired_lp" from="last_pass_value"/>
</MEM>
<OR>
    <!-- OR, AND, etc. Gates can specify any number of inputs via incrementing
         the input specifiers "input1", "input2", "input3", etc. -->
    <Connection to="input1" from="not_gate_output"/>
    <Connection to="input2" from="count_expired_lp"/>
    <Connection to="or_gate_output" from="output"/>
</OR>
<Constant name="timer_time" type=single value=10/>
<Subsystem name="timer_instance_1" reference="timer.vs">
    <Connection to="start" from="digital_input_1"/>
    <Connection to="reset" from="or_gate_output"/>
    <Connection to="time" from="timer_time"/>
    <Connection to="digital_output_1" from="count_expired"/>
    <!-- Any un-attached outputs are optimized out, e.g. elapsed_time -->
    <!-- All inputs are required -->
</Subsystem>
<Output scope=device name="digital_output_1" type=boolean>
    <!-- Hardware address 456 is DO_1 for ATMega328 -->
    <!-- TODO: Find real hardware address for this DO -->
    <Address>0x0456</Address>
</Output>
```

Listing 2: ../example/timed-blinking-light/timer.vs

```
<Style Tag>
<!-- All "parts" added by the user can use Inputs and/or
     Outputs for utilization elsewhere in project. The
     reference will search the path for that file -->
<!-- All Inputs do not have to be used and will be optimized out -->
<Input scope=local name="start" type=boolean/>
<Input scope=local name="reset" type=boolean/>
<Input scope=local name="time" type=single/>
<!-- All Outputs need to have a connection, at least to a constant -->
<Output scope=local name="count_expired" type=boolean/>
<Output scope=local name="elapsed_time" type=single/>
<Signal scope=local name="not_gate_output" type=boolean/>
<Signal scope=local name="and_gate_output" type=boolean/>
<Signal scope=local name="count_expired_lp" type=boolean/>
<Signal scope=local name="enable_state_output" type=single/>
<Signal scope=local name="reset_state_output" type=single/>
<Signal scope=local name="summer_output" type=single/>
<Signal scope=local name="elapsed_time_lp" type=single/>
<Signal scope=local name="time_since_last_pass" type=single/>
<Constant name="zero_constant" type=single value=0.000/>
<DT>
    <!-- The DT block puts out the difference in time between
         successive passes of program. In a Soft RTOS, this
         would be a variable number. In a Hard RTOS, this
         would be a constant number. -->
    <Connection to="time_since_last_pass" from="output">
</DT>
<NOT>
    <Connection to="input" from="count_expired_lp"/>
    <Connection to="not_gate_output" from="output"/>
</NOT>
<AND>
    <Connection to="input1" from="start"/>
    <Connection to="input2␣from="count_expired_lp"/>
␣␣␣␣<Connection␣to="and_gate_output"␣from="output"/>
</AND>
<IF>
␣␣␣␣<!--␣Control␣flow␣IF␣switch:␣If␣Control␣is␣true,␣execute
␣␣␣␣␣␣␣␣␣True␣assignment,␣else␣execute␣False␣assignment␣-->
␣␣␣␣<Connection␣to="control"␣from="and_gate_output"/>
␣␣␣␣<Connection␣to="true_input"␣from="time_since_last_pass"/>
␣␣␣␣<Connection␣to="false_input"␣from="zero_constant"/>
␣␣␣␣<Connection␣to="enable_state_output"␣from="output"/>
</IF>
<SUM>
```

```
␣␣␣␣<!--␣The␣summer␣will␣add␣all␣the␣inputs␣together.␣If␣you␣want
␣␣␣␣␣␣␣␣␣add␣a␣negative␣number,␣use␣the␣NEG␣part␣to␣negate␣the
␣␣␣␣␣␣␣␣␣signal␣before␣connecting␣to␣this␣part.␣-->
␣␣␣␣<!--␣Additionally,␣the␣PROD␣part␣exists␣for␣taking␣the␣PI
␣␣␣␣␣␣␣␣␣product␣of␣a␣set␣of␣inputs,␣and␣the␣INV␣command␣for␣taking
␣␣␣␣␣␣␣␣␣the␣recipicral␣of␣a␣number␣(divide␣by␣zero␣runtime␣error
␣␣␣␣␣␣␣␣␣possible)␣-->
␣␣␣␣<Connection␣to="input1"␣from="enable_state_output"/>
␣␣␣␣<Connection␣to="input2"␣from="elapsed_time_lp"/>
␣␣␣␣<Connection␣to="summer_output"␣from="output"/>
</SUM>
<IF>
␣␣␣␣<Connection␣to="control"␣from="reset"/>
␣␣␣␣<Connection␣to="true_input"␣from="zero_constant"/>
␣␣␣␣<Connection␣to="false_input"␣from="summer_output"/>
␣␣␣␣<Connection␣to="elapsed_time"␣from="output"/>
</IF>
<COMPARE␣operator=">=">
␣␣␣␣<Connection␣to="input1"␣from="elapsed_time"/>
␣␣␣␣<Connection␣to="input2"␣from="time"/>
␣␣␣␣<Connection␣to="count_expired"␣from="output"/>
</COMPARE>
<MEM␣initial_condition=0.000>
␣␣␣␣<Connection␣to="current_pass_value"␣from="elapsed_time"/>
␣␣␣␣<Connection␣to="elapsed_time_lp"␣from="last_pass_value"/>
</MEM>
<MEM␣initial_condition=0>
␣␣␣␣<Connection␣to="current_pass_value"␣from="count_expired"/>
␣␣␣␣<Connection␣to="count_expired_lp"␣from="last_pass_value"/>
</MEM>
```