Hardware Synthesis from SDF

• Start with SDF universe - abstract algorithm design
• Retarget to VHDL if corresponding stars available
• Target generates hardware blocks for individual functions
• Registers for state and for data transfer between blocks
• Initializers and selectors for registers
• Register updates between SDF schedule iterations
• Allows for resynthesis, rescheduling of hardware design

• Additional Goals
  • Feedback (back-annotate) info from synthesis to VHDL target
  • Regroup functional blocks to improve synthesis result
  • Generate control, clocks, resets automatically
New domain block style

- Stars written in terms of interface and function
- Macros similar to CGC, but elaborated differently
- Different code generated from same star by different VHDL targets

```vhdl
defstar {
    name { Ramp } 
    domain { VHDL } 
    output { name { output } } 
    defstate { name { step } } 
    defstate { name { value } } 
    codeblock (std) {
        $ref(output) $assign(output) $ref(value); 
        $ref(value) $assign(value) $ref(value) + $val(step); 
    } 
    go { 
        addCode(std); 
    } 
}
```
Future Release Plans

- No further additions to VHDLF, VHDLB domains

New VHDL Domain
- Fully inhomogeneous SDF semantics (multirate)
- Generate code in different styles depending on choice of target
  - Target to generate code for efficient simulation
  - Target to generate code for synthesis
  - Target to use with Synopsys synthesis tools
  - Target to use with Synopsys simulator
- Co-simulation between Ptolemy and Synopsys
- FPGA synthesis and download to target architecture on Sparcstation bus board for rapid prototyping
Some VHDL Demos

VHDLF Demos
The VHDLF domain is an experimental, and still underdeveloped domain for VHDL code generation for functional blocks with SDF-like semantics.

VHDLB Demos
The VHDLB domain is an experimental, and still underdeveloped domain for VHDL code generation for behavioral blocks with DE-like semantics.
Some VHDL Stars
Steps in VHDL Code Generation

- VHDLTarget iterates through all blocks
- For each block, iterates through all ports
- Constructs top-level entity
- Instantiates an entity for each star in the graph
- Instantiates a signal for each arc
- Specifies port connections accordingly
- Generates code hierarchically for Ptolemy galaxies
- Configuration references to the VHDL code in the library directories
• Corresponding VHDL code file Vhdlfxxx.vhdl
• In $PTOLEMY/src/domains/vhdlf/lib/ directory
• Entity, architecture, and empty configuration

entity VHDLFNot is
    port (input: in integer; output: out integer);
end VHDLFNot;

architecture VHDLFNot_behavior of VHDLFNot is
begin
    process
    begin
        wait on input’transaction;
        if input /= 0 then
            output <= 0;
        else
            output <= 1;
        end if;
    end process;
end VHDLFNot_behavior;
• Star definition file VHDLFxxx.pl
• In $PTOLEMY/src/domains/vhdlf/stars/ directory
• Describes only the interface: ports, states, datatypes
• Empty go() method

```vhdl
defstar {
    name { Not }
    domain { VHDLF }
    input {
        name { input }
        type { int }
    }
    output {
        name { output }
        type { int }
    }
    go {
    }
}
```
VHDL Block Definition Organization

- Block writer ensures correspondence between the two
- Import existing VHDL code module by writing a star definition for it
Two VHDL Domains in Ptolemy

VHDL-Functional (VHDLF)
- No timing dependency
- Actors compute fixed functions on data streams
- Computations occur at discrete intervals
- Computations occur with zero latency
- Synchronous Dataflow (SDF)-like semantics

VHDL-Behavioral (VHDLB)
- Timing dependency
- Actors may have time-dependent behavior
- Computations may occur at any time
- Computations may have non-zero latency
- Discrete Event (DE)-like semantics
Some CGC Demos

- CGC Demos
  - C-code generation demos for single processor and multiple processors
  - Basic
  - Multirate
  - Signal Processing
  - Multi-Processor Demos
  - Fixed-Point Demos
  - Sound
  - Tcl/Tk
  - BDF
  - Higher Order Functions

- Basic demos illustrating simple uses of Ptolemy and the use of certain stars

- The Butterfly Curve
Some CGC Stars

- Signal Sources
- Signal Sinks
- Arithmetic
- Nonlinear Functions
- Control
- Conversion
- Signal Processing
- Communications
- CGC/BDF stars
- Tcl/Tk stars
- Higher Order

CGC Signal Sources

- Fixed-Point Sources
  - CGC-specific

Signal Processing
Tcl/Tk in CGC

- Tcl/Tk stars in CGC, comparable to Tcl/Tk stars in SDF
  - TclTk_Target

Additional Methods for TclTk stars in CGC

- void errorReport()
- void makeEntry()
- void makeButton()
- void makeScale()
- void displaySliderValue()
BDF in CGC

- Use bdf-CGC target
- Two principal stars: switch and select
- `setRelation()` method
  - DF_NONE
  - DF_TRUE
  - DF_FALSE
  - DFSAME
  - DF_COMPLEMENT

```c
trueOutput.setRelation(DF_TRUE, &control);
falseOutput.setRelation(DF_FALSE, &control);
```
Buffer Embedding

- Data communication across arcs implemented using buffers
- Avoid allocating buffers for certain stars: spread, collect, fork, upsample, downsample (use embedding of static buffers if possible)
- Declare buffer embedding in the setup method

```java
setup {
    input.embed(trueOutput, 0);
    input.embed(falseOutput, 0);
}
```

![Diagram](https://via.placeholder.com/150)

- no embedding
- embedding
**CGC `addCode` Methods and Streams**

Additional methods to add to specific code streams
- `addInclude()`
- `addDeclaration()`
- `addGlobal()`

Additional code streams
- `CodeStream include`
- `CodeStream mainDecls`
- `CodeStream globalDecls`
- `CodeStream mainInit`
- `CodeStream mainClose`
for (i_3=0; i_3 < 10; i_3++) {
    {  /* star SmallTest.Ramp1 (class CGCRamp) */
        output_0 = value_4;
        value_4 += 1.0;
    }
    {  /* star SmallTest.XMgraph1 (class CGCXMgraph) */
        if (++count_5 >= 0) {
            fprintf(fp_1[0],”%g %g
”,index_6,output_0);
        }
        index_6 += 1.0;
    }
} /* end repeat, depth 0*/

{ int i;
    for (i = 0; i < 1; i++) fclose(fp_1[i]);
    system("( pxgraph -t ‘X graph’ -bb -tk =800x400 default-CGC_temp_20 ; /bin/rm -f default-CGC_temp_20) &”);
}

#include <stdio.h>

/* main function */
main() {
    FILE* fp_1[1];
    int i_3;
    double value_4;
    double output_0;
    int count_5;
    double index_6;
    count_5=0;
    index_6=0.0;
    value_4=0.0;
    output_0 = 0.0;
    if(! (fp_1[0] = fopen("default-CGC_temp_20","w")))
    {
        fprintf(stderr,"ERROR: cannot open file.\n");
    }
    for (i_3=0; i_3 < 10; i_3++) {
        /* star SmallTest.Ramp1 (class CGCRamp) */
        output_0 = value_4;
    }
A Small Test System
Generating Code for a Single Firing

Adding one firing’s code to the code stream

```go
{  
    addCode(std);  
}
...

codeblock (std) {  
    $ref(output) = $ref(value);  
    $ref(value) += $val(step);  
}
```

An alternative way to do the same thing

```go
{  
    StringList out;  
    out << "\t$ref(output) = $ref(value);\n";  
    out << "\t$ref(value) += $val(step);\n";  
    addCode((const char*)out);  
    out.initialize();  
}  
```
...  
defstate {
    name { step }
    type { float }
    default { 1.0 }
    desc { Increment from one sample to the next. }  
}
defstate {
    name { value }
    type { float }
    default { 0.0 }
    desc { Initial (or latest) value output by Ramp. }  
    attributes { A_SETTABLE|A_NONCONSTANT }  
}
...

- States will correspond to resources in generated code  
- Attributes guide the target in deciding how to allocate resources
defstar {
    name { Ramp } 
    domain { CGC } 
    desc {
Generates a ramp signal, starting at "value" (default 0) 
with step size "step" (default 1).
    }
    author { E. A. Lee } 
output {
    name { output } 
    type { float } 
}
defstate {
    name { step } 
    type { float } 
    default { 1.0 } 
    desc { Increment from one sample to the next. } 
}
defstate {
    name { value } 
    type { float } 
}
Writing CGC Blocks

- Each block has a star definition file
- Similar to simulation blocks in overall structure
- Additional special methods for CG and for derived domains
- Essential elements:
  - name, domain
  - inputs, outputs, state
  - codeblocks
  - go method
Outline

• CGC
  • Writing CGC Blocks
  • Example Star
  • CGC addCode Methods and Streams
  • Buffer Embedding
  • BDF in CGC
  • Tcl/Tk in CGC

• VHDL
  • Two VHDL Domains in Ptolemy
  • VHDL Block Definition Organization
  • Writing VHDL Blocks
  • Steps in VHDL Code Generation
  • Future Release Plans
  • New Domain Block Style
  • Hardware Synthesis from SDF
Writing Blocks for CGC and VHDL

Michael C. Williamson

UC Berkeley