Code Generation Concepts

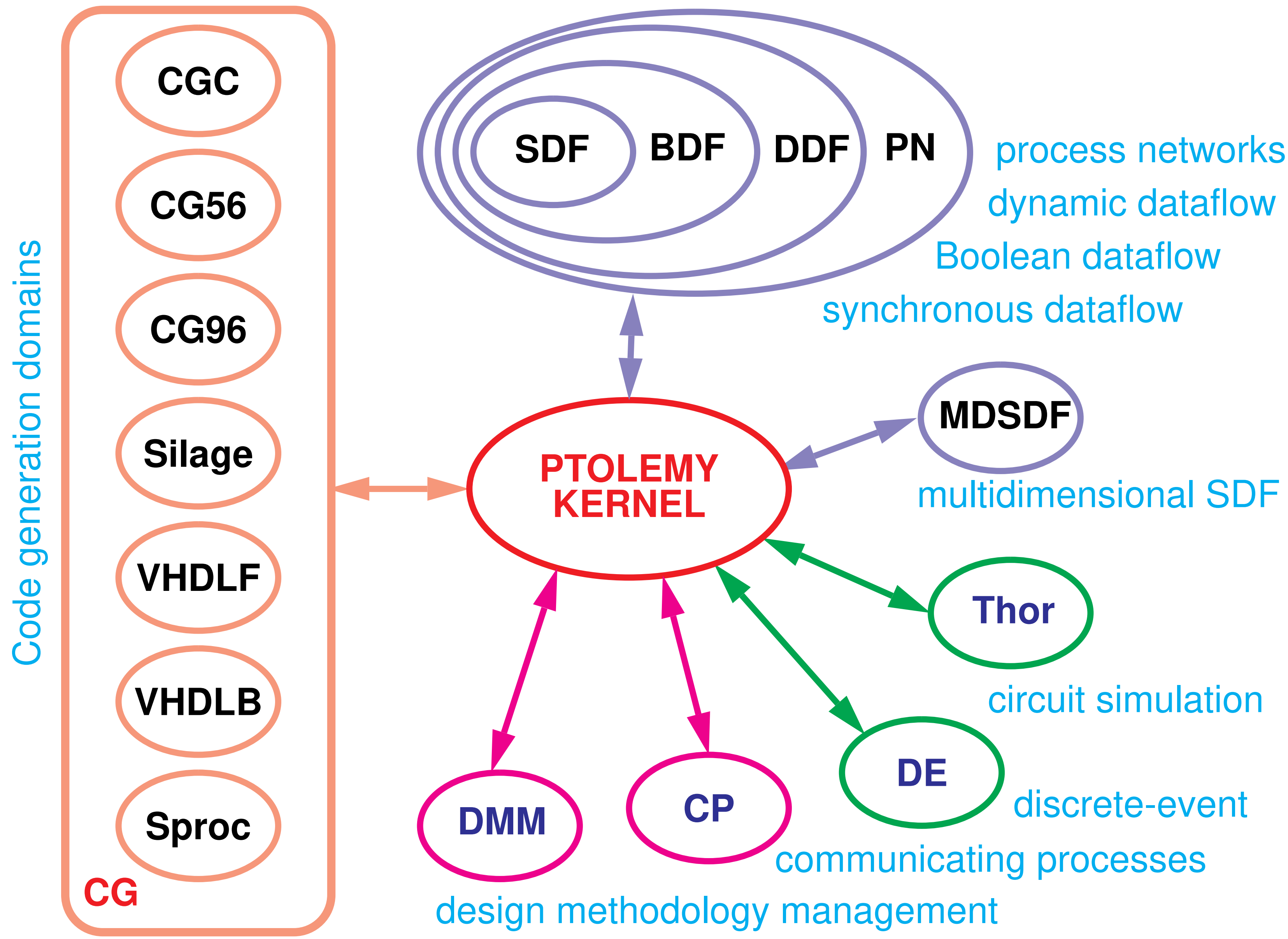
Michael C. Williamson

UC Berkeley

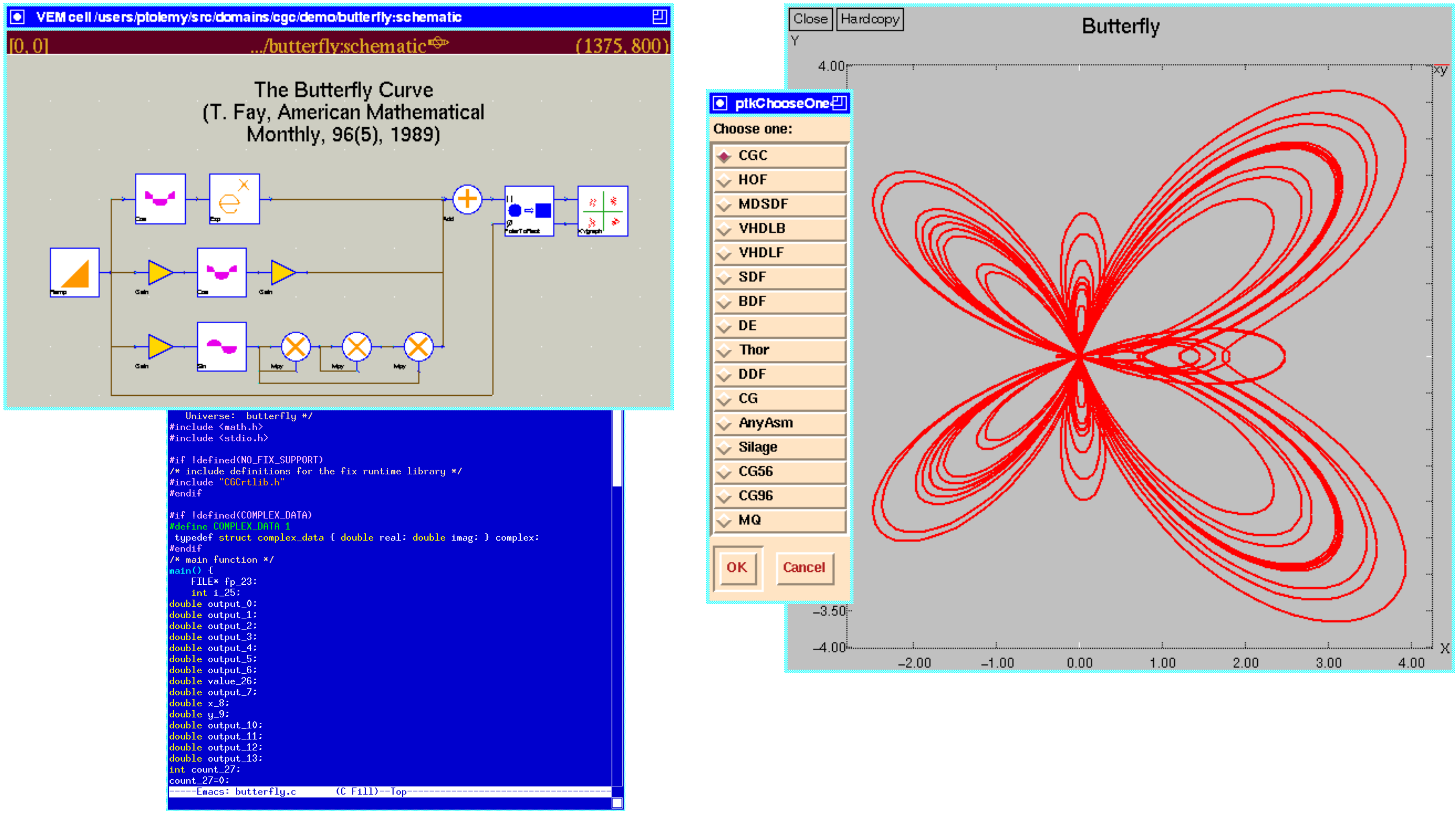
Outline

- Applications of Code Generation
- Relationships Between Domains
- Code Generation Classes
- Elements of Code Generation Stars
  - Codeblocks
  - Code Streams
  - Macros
  - Attributes
- Code Generation Targets
- Multitargets
- Methods in Code Generation
- Schedulers and Code Size

Domains in Ptolemy

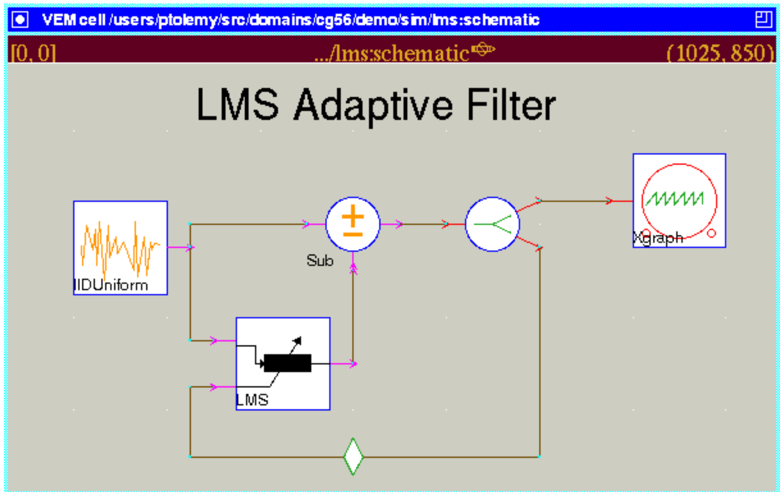


Generate Standalone Simulation Programs

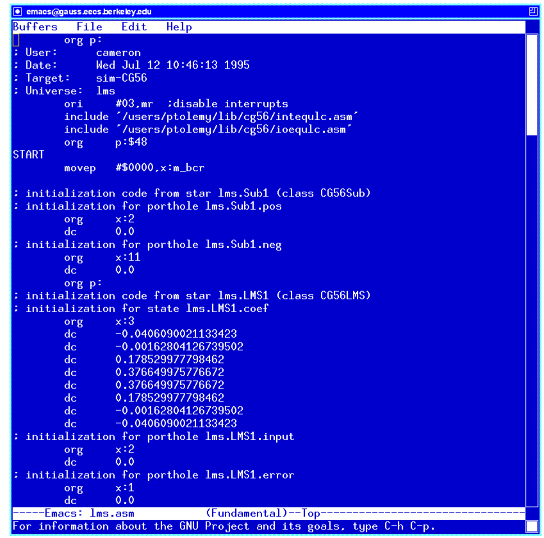


- **Retarget from SDF or build from scratch in CGXXX**
- **Run on workstation independent of Ptolemy**
- **Recompile code to run on other platforms**

Generate Code for Special-Purpose Processors



Generated Code



```
emacs@gauss.eecs.berkeley.edu
Buffers   File   Edit   Help
        org p:
; User:       cameron
; Date:       Wed Jul 12 10:46:13 1995
; Target:     sim-CG56
; Universe:   lms
        ori     #03,mr  ;disable interrupts
        include '/users/ptolemy/lib/cg56/intequlc.asm'
        include '/users/ptolemy/lib/cg56/ioequlc.asm'
        org     p:$48
START
        movep   #$0000,x:m_bcr

; initialization code from star lms.Sub1 (class CG56Sub)
; initialization for porthole lms.Sub1.pos
        org     x:2
        dc      0.0
; initialization for porthole lms.Sub1.neg
        org     x:11
        dc      0.0
        org p:
; initialization code from star lms.LMS1 (class CG56LMS)
; initialization for state lms.LMS1.coef
        org     x:3
        dc      -0.0406090021133423
        dc      -0.00162804126739502
        dc      0.178529977798462
        dc      0.376649975776672
        dc      0.376649975776672
        dc      0.178529977798462
        dc      -0.00162804126739502
        dc      -0.0406090021133423
; initialization for porthole lms.LMS1.input
        org     x:2
        dc      0.0
; initialization for porthole lms.LMS1.error
        org     x:1
        dc      0.0
-----Emacs: lms.asm          (Fundamental)--Top-----------------------------
For information about the GNU Project and its goals, type C-h C-p.
```

Compiler

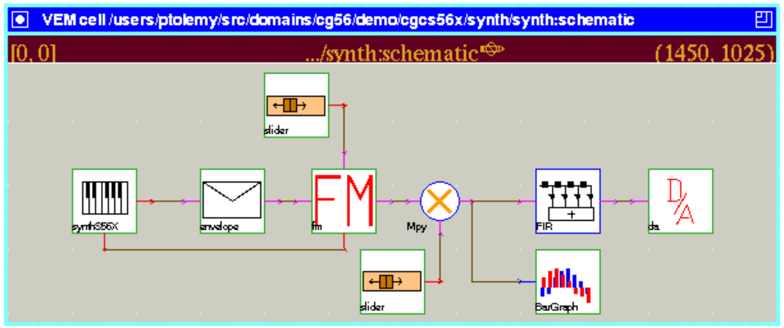
Processor Simulator

Processor Architecture

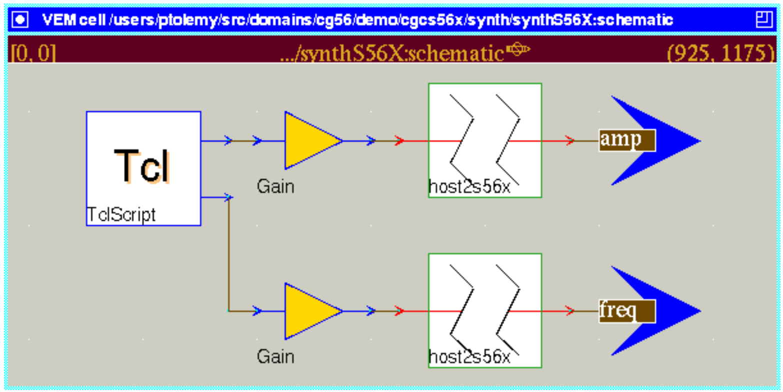
- **Domains: CG56, CG96, Sproc**

Generate Code for Heterogeneous Architectures

Top-level CG56 system



CGC subsystem



56000 Code

Run on processor on S-56X card

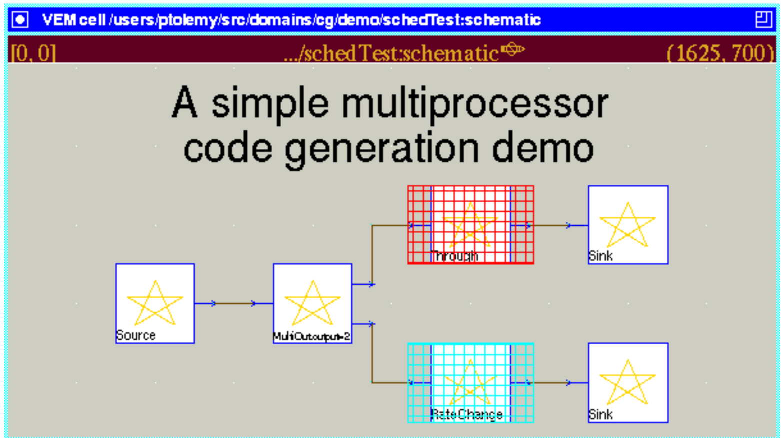
C Code

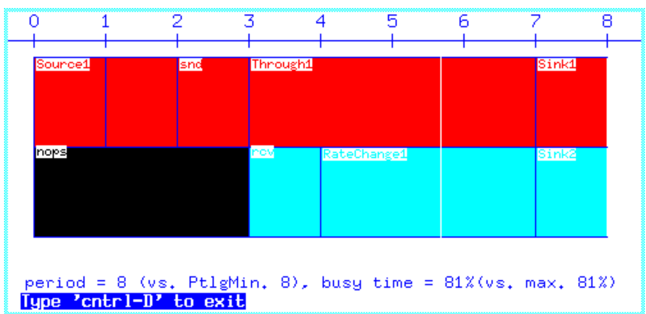
Run on SparcStation

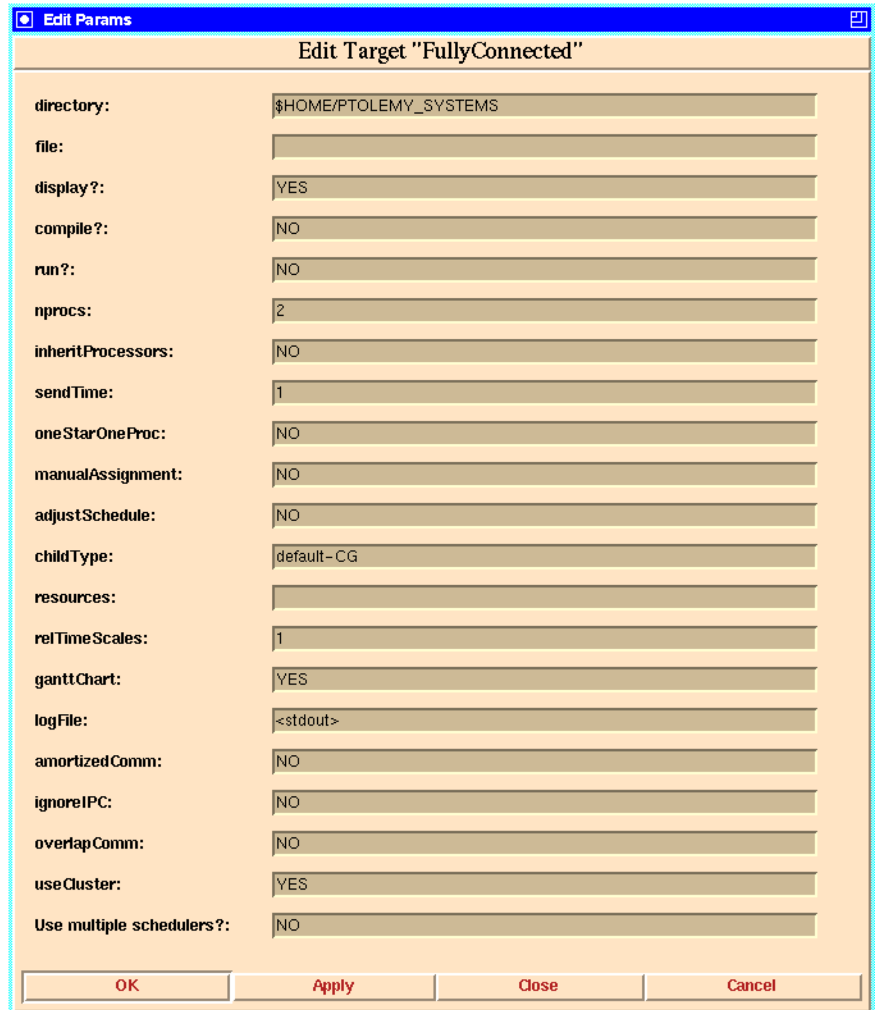
Target sets up communication

- Domains: CG56
- Targets: CGC-S56X

Test Multiprocessor Schedulers

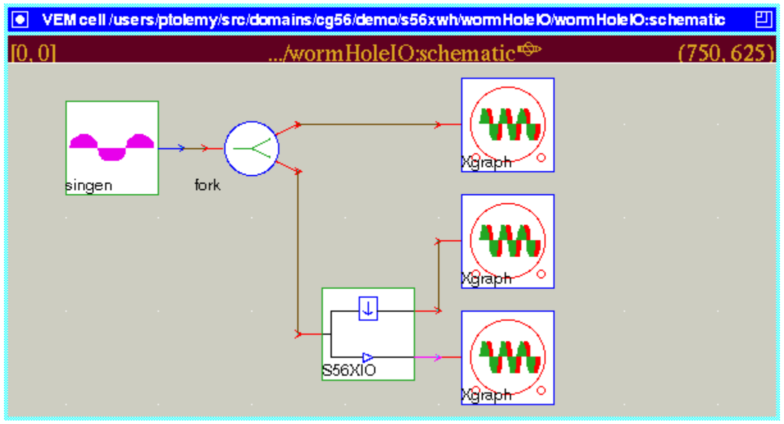




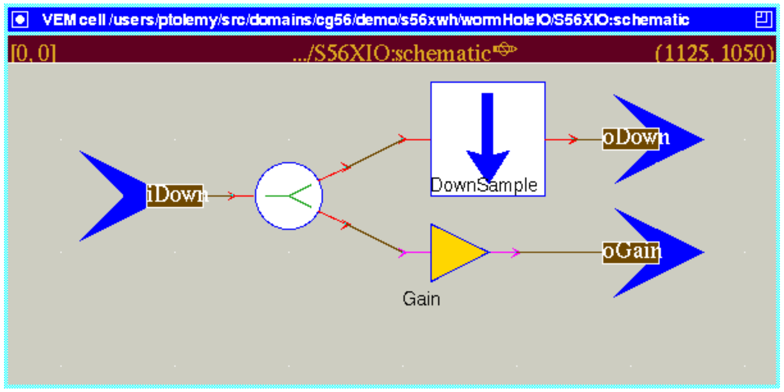


- **Domains: CG; CGTargets: FullyConnected, SharedBus**
- **Other Targets: unixMulti_C, MultiSim-56000**

Generate Code for Co-simulation



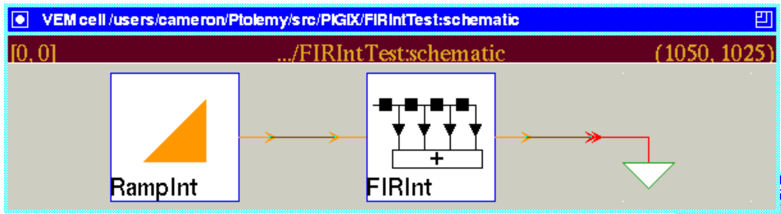
Top-level SDF system

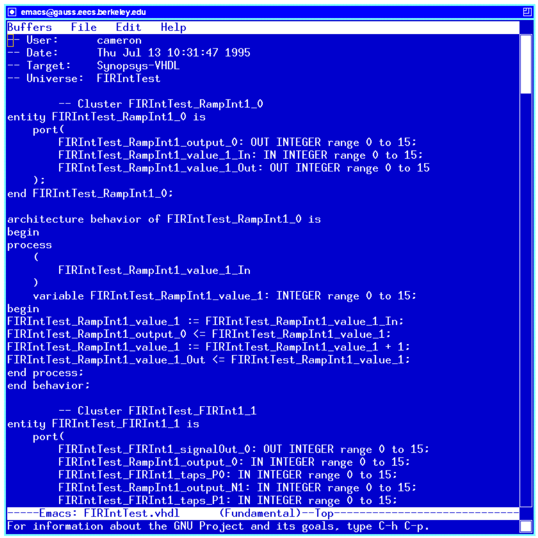


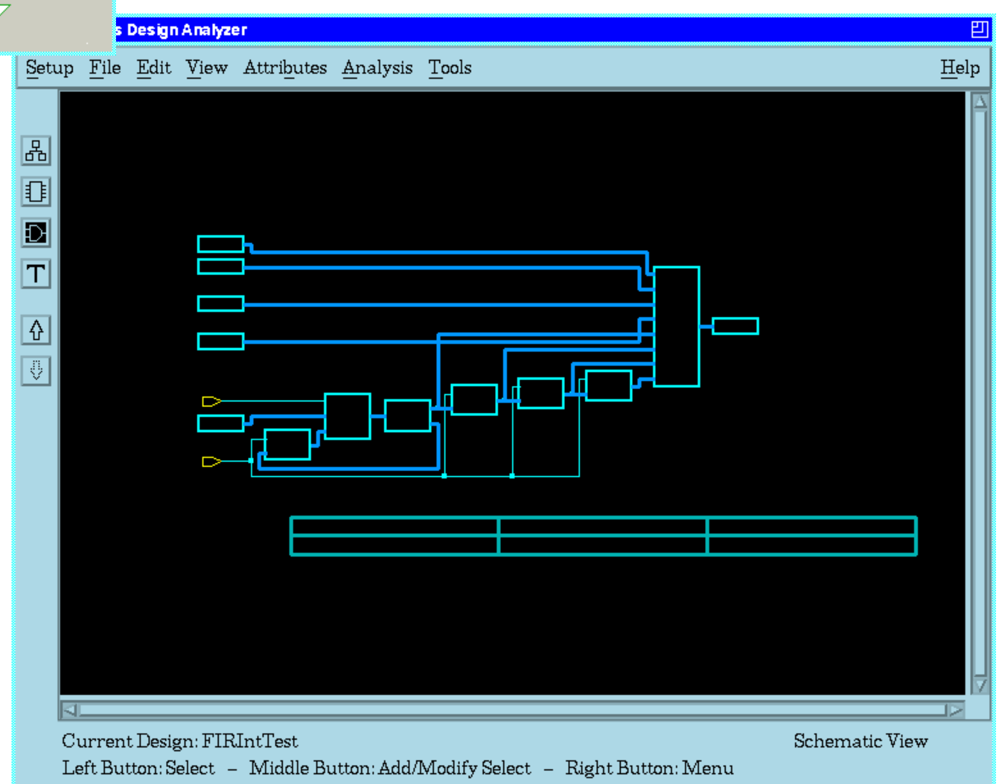
CG56 subsystem

- Generate code for co-simulation between Ptolemy and code running on hardware connected to a workstation
- Targets: S-56X, S-56XWH

Generate System Descriptions for Design Tools







- Generate system descriptions for processing by other design tools (Silage-->Hyper, VHDL-->Synopsys)
- Domains: Silage, VHDLF, VHDLB

Relationships Between Domains

Simulation Domains ←- - - - -→ **Code Generation Domains**

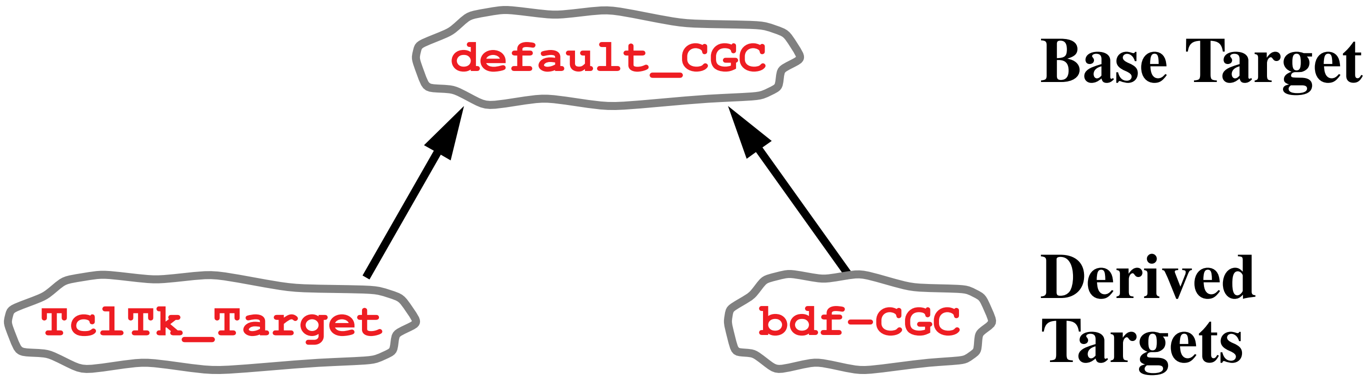
Retargetting

Modeling and Simulation

Code Generation and Compiled Simulation

- CG domain for comment generation, test schedulers
- Domains derived from CG for language-specific code generation
- Dataflow semantics supported, other semantics possible
- CG kernel: processor-independent actions
  - allocate memory for buffers, constants, tables
  - generate symbols, labels

Code Generation Classes



- **Domains correspond to particular languages (C, assembly, Silage, VHDL)**
  - Default target
  - Supported star type
- **Targets correspond to different target architectures**
  - Derived from base class `target`
  - Control scheduling, compiling, assembling, downloading code
  - Multiprocessor architectures: task partitioning and synchronization

Elements of Code Generation Stars

- Similar to simulation stars: `setup()`, `go()`, `wrapup()` methods
- Additional methods: `initCode()`, `execTime()`

- `setup()`: Initialize local variables and states, called before schedule is generated
- `initCode()`: Generate initialization code before the main loop, procedure declaration code outside the main loop, called after schedule is generated
- `go()`: Generate code in the main loop for one firing
- `wrapup()`: Generate code after the main loop
- `int execTime()`: Returns execution time of one firing in main loop in processor cycles or instruction steps; Used primarily in parallel scheduling

Codeblocks

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

go { addCode(std); }
```

- A section of code which can be generated by a star
- Written in the output code language with interspersed macros to be expanded by preprocessing
- Referred to by an identifying name (`std`)
- Added to output code with `addCode()` or related methods
- Decision-making during star firing for conditional code generation

Code Streams

- Distinguish among different sections of code (initialization, main loop, wrapup, declarations, procedures, functions, compiler directives)
- Methods related to `addCode()` to add code to specific code streams (e.g. `addProcedure()` to add code to `procedures` code stream)
- Variations of `addCode()` with arguments to control adding code to an argument code stream
- Check if same code has already been added, avoid redundant code for shared resources

Macros

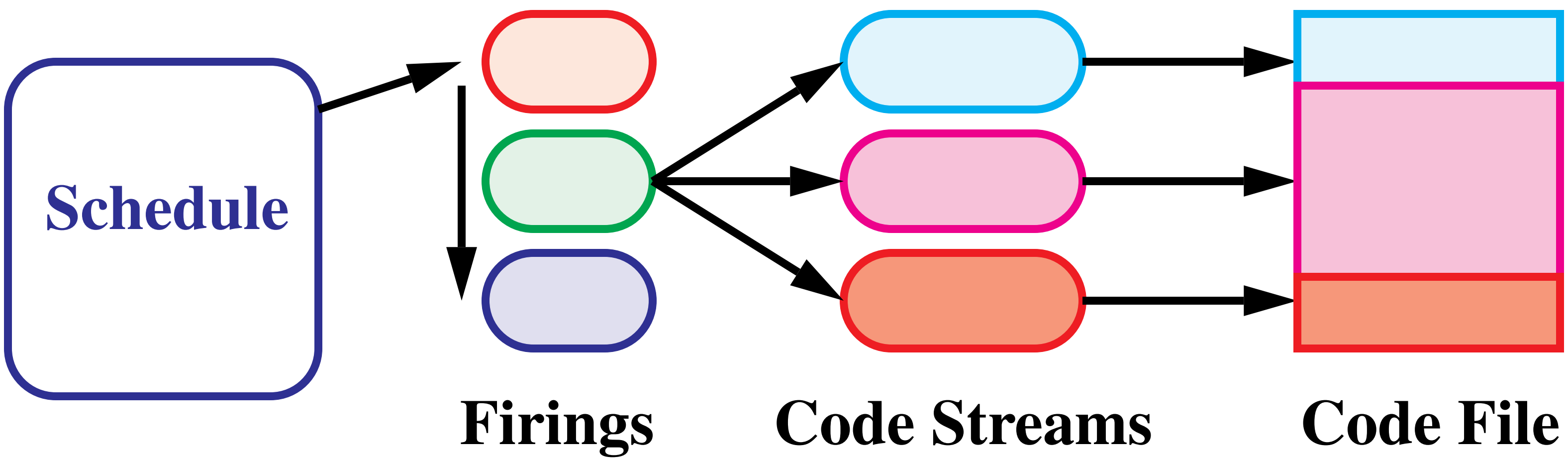
- **Base CGStar Macros**
  - `$ref(name), $ref(name, offset)`
  - `$val(state-name)`
  - `$size(name)`
  - `$starName()`
  - `$fullName()`
  - `$starSymbol(name)`
  - `$sharedSymbol(list, name)`
  - `$label(name), $codeblockSymbol(name)`
- **Additional Macros**
  - `$addr(name), $addr(name, offset)`
  - `$ref(name, offset)`
  - `$mem(name)`

Attributes

- **State Attributes**
  - `A_GLOBAL, A_LOCAL`
  - `A_SHARED, A_PRIVATE`
  - `A_CONSTANT, A_NONCONSTANT`
  - `A_SETTABLE, A_NONSETTABLE`
  - `A_CIRC`
  - `A_CONSEC`
  - `A_MEMORY`
  - `A_NOINT`
  - `A_REVERSE`
  - `A_SYMMETRIC`
  - `A_ROM, A_RAM`
- **Porthole Attributes**
  - `P_CIRC, P_SHARED, P_SYMMETRIC, P_NOINT`

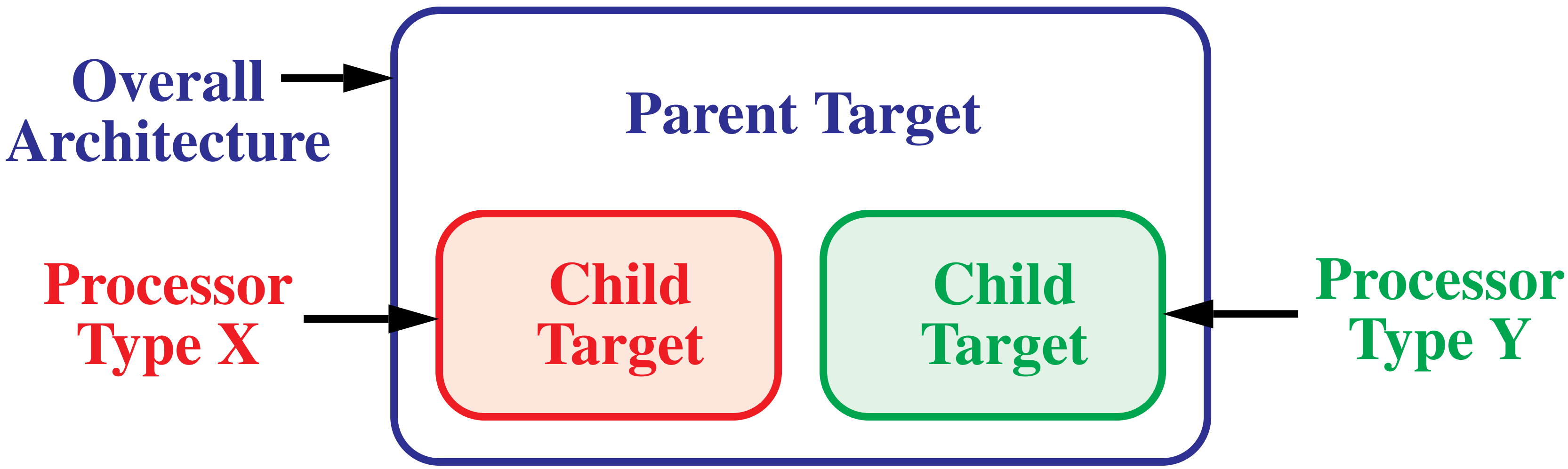
Code Generation Targets

- Targets correspond to architectures (single processor, multiprocessor, fully connected, shared bus, etc.)
- Multiple targets available for a single code generation domain
- Methods to generate schedule, fire individual stars
- Methods to generate individual code streams, assemble code streams into code file, generate code file
- Methods for compiling code, downloading code, running code



Multitargets

- Multiprocessor architectures
- Use targets hierarchically
- Heterogeneous architectures use heterogeneous targets
- Child targets generate code for their portion of the application
- Parent target coordinates all code generation and interaction with the architecture
- Special stars for synchronization between processors



Methods in Code Generation

- `generateCode()`
- `allocateMemory()`
- `codeGenInit()`
- `mainLoopCode()`
- `go()`
- `wrapup()`
- `frameCode()`
- `writeCode()`
- `compileCode()`
- `loadCode()`
- `runCode()`

Schedulers and Code Size

- For SDF semantics, use Ptolemy's single-processor SDF schedulers (default, Joe's, SJS) and multi-processor SDF schedulers (Hu level, Sih dynamic level, Sih declustering)
- Code size varies dramatically for different schedulers, especially for systems with large sample rate changes
- Buffer size also varies with different schedulers
- Looping requires indirect addressing of buffers, more overhead (buffer pointer, indirect addressing)
- Tradeoff code size vs. buffer size

```
/* Schedule 1 */

fire_star_A();
fire_star_A();
fire_star_A();
fire_star_A();
fire_star_A();
```

```
/* Schedule 2 */

repeat(5) {
    fire_star_A();
}
```