Graphical System Design

David Fuller
LabVIEW R&D Section Manager
Agenda

• Visions
• Demo
• Order & time
Virtual Instrumentation
Virtual Instrumentation
LabVIEW Virtual Instrument

Front Panel

Block Diagram
The G (LabVIEW) Language Model

- Homogenous dataflow language
  - Structured case (switch, select) and loops
    - “Structured dataflow”
- Run-time scheduling
  - Explicit task level parallelism
  - Implicit parallelism heuristically identified
- Synthesizable language
  - To machine code on x86 and PPC processors
  - To VHDL for FPGAs
  - To C for embedded processors
- Turing complete
Graphical System Design

“To do for embedded what the PC did for the desktop.”
Graphical System Design (cont’d)
System Platform Stack

A. Sangiovanni-Vincentelli, UC Berkeley. Defining Platform Based Design. EEDesign, Feb 2002
Graphical System Design Targets

- Embedded Controllers
- HMI and PDA
- PXI Industrial Controllers
- FPGAs on NI RIO
- Sensors
- Microprocessors
- Vision Systems
- Embedded Controllers
- Programmable Automation Controllers
- Portable PCs
- Desktop PCs
Graphical System Design Synthesis

LabVIEW Embedded Technology

- Native Compiler
- VHDL Generation
- C Code Generation
- Real-Time Controller
- FPGA
- Industrial HMI
- PocketPC Device
- Any 32-bit MPU
Vision Implications

• Processors & FPGAs are central to our platform
• VIs are NI’s common software representation
  – VIs run on all our platform targets
• NI supplies rich domain specific libraries
  – For test, control and design
• VIs are portable across targets
  – IO and timing are abstracted to a degree
Abstraction Paradox

- Abstraction tends to increase portability
- Abstraction tends to increase efficiency
- Abstraction tends to hide complexity

Unfortunately
- One person’s abstraction is another's obstruction
- Abstraction tends to introduce *leaky abstractions*
Abstraction Paradox Example
Abstraction - Genius of the And

• Increase logic portability and development efficiency with high-level abstractions

AND

• Create hierarchies of abstractions enabling users to decide at which level they will work

• Implies higher-level abstractions are composed from elemental building blocks
Philosophy of *LabVIEW to the Pin*

- A highly resource aware language & environment
  - Convey full richness of architectural space visually
- Minimum number of the right abstractions
  - Elemental building blocks model physical world closely
- Minimal opaque or uncontrollable side-effects
  - Elemental building blocks don’t leak
- APIs feel like language extensions, not APIs
LabVIEW to the Pin

BLUEPRINT DEMO
Automatic/Manual – Genius of the And

• Graphically map logic to richly visualized hardware resources with wiring metaphor
  – We view manual control and exploration as an invariant
  – Aid user in manual mapping through editor
    • Track mapping points; provide batch mapping/remapping

AND

• Automate the mapping of logic to hardware
  – Optimal assignment of logic to processing elements
  – Optimal utilization of network and IO
TIME AND ORDER
G Dataflow  (Structured homogenous dataflow)

A → B → C

Produces Order
G Dataflow with Structures

Produces Order
G Dataflow is Intuitive and Simple

- Diagram representation matches human understanding of the relationship of space to order
  - A precedes B precedes C
    - assuming left-to-right mindset
- Data exchange matches human understanding of physical existence
  - A datum is active in but one “place” at one time
  - Thus, it flows from one actor to another like a baton
G Dataflow Wires Cannot Represent…

• Multi-rate Actors
  – Actors with complex data and execution relationships beyond a datum active at one place at one time
    • Such as pipelined actors

• Data-Independent Actors
  – Actors with order or time relationships only
Jeff’s Wire  *(amorphous heterogeneous dataflow)*

- The most basic notion of a kind of wire
  - Wires have direction
  - No semantic constraints
- Represents any relationship
  - “There exists a relationship between…”
Jeff’s Wire (cont’d)

A Possible Order

Another Possible Order

Many more… Function of wires, A, B, and C

Order could be anything
Jeff’s Wire (cont’d)

• Execution order is a function of the actors and the data exchange policies of wires
  – Order is not inferable from looking at the diagram

• Diagrams with Jeff’s wires will tend to be more complex than traditional G diagrams
Jeff’s Wire and Multi-Rate

• Multi-rate
  – Actors may execute concurrently with different rates
  – Two views on rate
    • Rate == execution order ratios (block B runs 3 times to 1 versus A)
    • Rate == period of execution
  – Jeff’s wire enables rate control through execution order ratios

• Multi-rate Examples
  – Distributed system (PC to embedded to FPGA)
  – IO to software
  – Two free-running actors
  – Pipelined computation (streaming/DSP)
Jeff’s Wire – Standard Data Exchange

• With VIs, Jeff’s wire can have any behavior
• But, we also plan to provide default behaviors…
  – Buffers, FIFOs, queues, mailboxes and so on
  – Blocking and non-blocking policies
  – Behaviors MUST be common regardless of execution target and communication bus
• This matches general Ptolemy notion of rate
  – rate derived from data exchange order
Jeff’s Wire – Remaining Gaps

• Observability of data and execution order
  – Need to monitor or watch data exchange
    • Visualize FIFOs, Queues and so on
  – Need tools to monitor execution order

• Debugging
  – Need to support single-step and other notions within context of a given set of multi-rate actors

• Startup and shutdown
  – For pipelining & other complex execution/exchanges
But what about Time?

What is time?

Or

What is time not?
Time Is Not Order

• With G dataflow, users specify data relationships and LabVIEW produces an **execution order**

• Execution order describes actor relationships
  – A precedes B precedes C

• Execution order is not actively related to Time
  – Other than a “before and after” notion

• But…
Time Does Relate to Order

• User apps may ignore Time and still observe execution order in Time
  – This is a passive relationship with Time
• User’s apps often need to actively relate execution order to Time
Synchronization

• Synchronization is the active coordination of actors with real world phenomena including Time
• User’s synchronize actors to the world with
  – Events (triggers)
  – Clocks (periodic events; clock ticks)
  – Time (standardized periodic events; clock ticks)
    • Apparent time, mean time, Greenwich time
Synchronization (cont’d)

• Actors wait on triggers, clocks, and Time to synchronize with the real world
  – An actor waits on a trigger
  – An actor waits on a clock tick
  – An actor waits on a Time clock to tick
• Thus, G Dataflow waits on Time
Time and Order

• Hypothesis: LabVIEW can fully synchronize Time & Order
• But, our representation and tools are data-oriented, not yet time oriented
Time and Order – Relationships

Absolute Time Line

Relative Time Line

Events

Clock tick

t_0

clock tick

A

B

C
Time and Order

- Constrain - This should take a second
- Observe – How long did this take?
- Predict - How long will this take?
- Schedule – Run this once a second
Constrain Order in Time

• Define constraints (relationships) between actors and absolute and relative timelines and events
  – Predict if execution will satisfy constraints
  – Observe if execution satisfies constraints
  – React if execution does not satisfy constraints
Time and Order – Example Constraints

- Start
- Tolerance
- Delay
- Deadline

A B C

\( t_0 \)
Observe Order in Time

• Observe the relationship of actors to absolute and relative timelines and events
  – Observe during run-time
  – Record and review after run-time
LabVIEW Trace Tool
Predict Order in Time

- Predict, at edit-time, the relationship of actors to absolute and relative timelines and events
  - Assists rapid application development when time is a first-order problem
  - Necessary, but not sufficient, for timing related environment feedback and compiler optimizations
- Such as longest path, meeting timing for FPGA, clues for compiler to pipeline and so on
Schedule Order in Time

• Define when actors will execute relative to absolute and relative and events
  – Predict if the schedule can be satisfied
  – Observe if the schedule is satisfied
  – React if the schedule is not satisfied

• Closely related to constraints
  – Related vocabulary; mirror images
  – With constraints, dataflow dominates execution order and that order is observed against time
  – With schedules, time dominates execution order
Schedule Order in Time (cont’d)

• Wikipidia on synchronization (or scheduling)

• “Whilst well-designed time synchronization is an important tool for creating reliable systems, excessive use of synchronization where it is not necessary can make systems less fault-tolerant, and hence less reliable.”
Time and Order – Scheduling Examples

\[ t_0 \]

\[ \text{start} \]

\[ \text{delay} \]
Graphical System Design - Summary

• We are creating our next-generation language and hardware with which domain experts will…
  – graphically describe, explore and visualize FPGA and processor-based embedded systems
  – graphically program with their choice of model of computation and abstraction level
  – graphically relate order & time
• We see strong vision alignment between Berkeley and National Instruments
NI’s Areas of Interest with Berkeley

- Ptolemy and Metroll (Modeling)
- PRET and PTIDES (Time and order)
- MESCAL (Mapping to HW Architectures)
- ParLab (Many core, Concurrency)
- Embedded Use Cases

- Opportunities for joint research...
QUESTIONS