

VSIPPL++ / FPGA Design Methodology

Capt. Jules Bergmann, AFRL/IFTC
Susan Emeny, ITT
Peter Bronowicz, ITT

Overview



- Introduction- Designing for Hybrid Architectures
- Design Methodology
- VSIPL++ / FPGA Integration
- Integration and Test
- Applications
- Status
- Future Work
- Conclusions

Introduction



- Hybrid Computer Architectures
 - FPGAs and Programmable Processors have the potential to deliver high performance
- Commercially Available
- Challenge of hybrid architectures to develop a methodology that will:
 - Exploit their capabilities effectively while
 - Making FPGAs accessible to a larger community of developers.

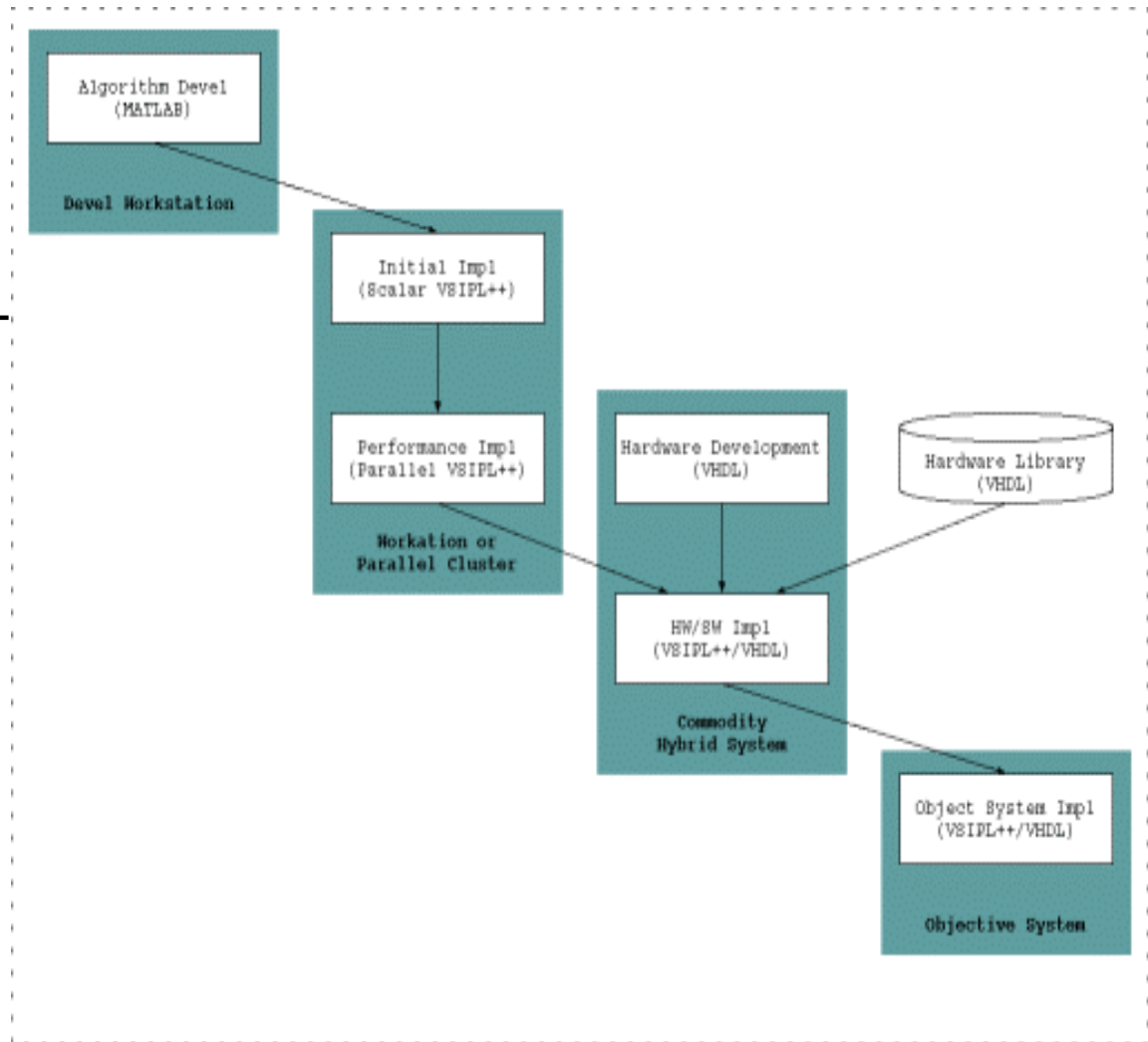
Requirements



Requirements of the Methodology

- Hardware & Software development needs to begin early
- Portable from test to final system, minimal change
- Scalable to future technologies, minimal change
- Productive
 - Concise function description for both HW & SW
 - Streamlined interface between HW & SW
- Use **existing** hardware and software methodologies

- **Algorithm**
 - High level exploration (Matlab)
- **Software**
 - Scalar, C++, VSIPL++
 - Performance Imp. (parallel)
- **Hardware**
 - VHDL
 - FPGA Performance libraries
- **Integration**
 - HW/SW Debug on commodity cluster
 - Migrate to target system



Benefits of the Methodology

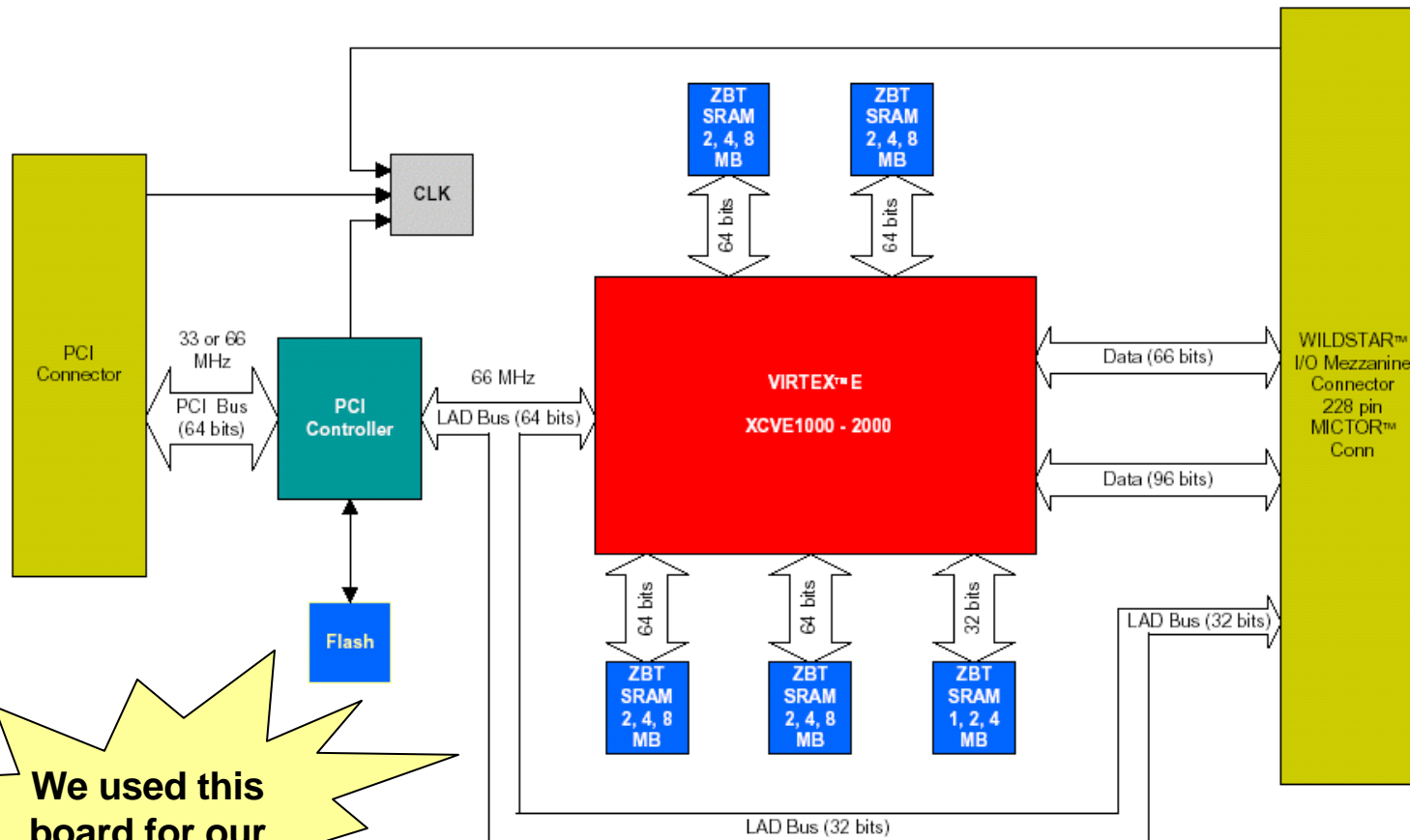


- Support for System Design from algorithm to embedded system
- Simplifies integration of hardware and software
- Application acceleration via pre-defined FPGA libraries
 - Standard functions (fft)
 - Custom

Hardware Design Methodology



- Portability & Scalability
 - Use standard high level synthesis (RTL)
- Performance
 - Encapsulate device specific optimizations in macro cells
- Productivity
 - Standardize interface between units for data exchange
 - Use stream interface with flow control. This model matches the way data is usually produced from sensors and requires minimal assumptions about environment



We used this board for our 1st Prototype!

Annapolis Firebird PCI Board

Software Design Methodology



VSIPL++ was chosen to achieve:

- Portability, the reference version compiles anywhere!
- Scalability, builds on existing standards i.e.. MPI
- Performance
 - Allows for optimized libraries which take advantage of specific machine features (transparent to application)
 - Allows for user defined functions (i.e. specialized or optimized FPGA functions)
- Productivity
 - Express computation in a natural fashion

Co-Design Issue



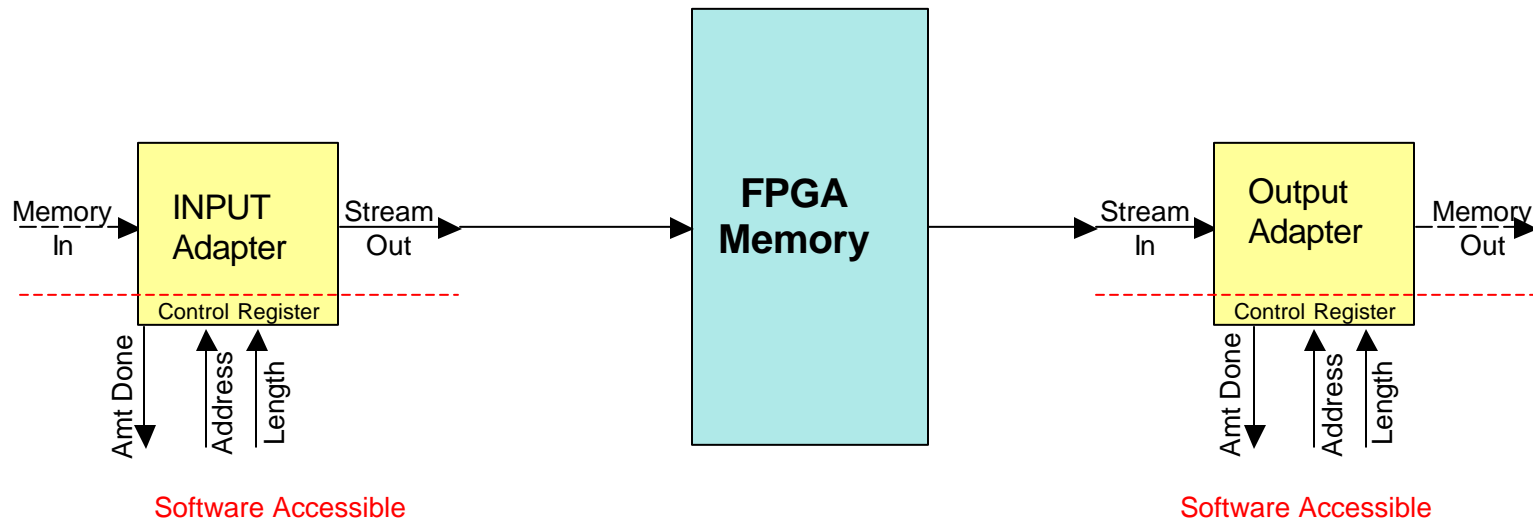
- VSIPL++ for software & Synthesizable compose-able modules for hardware are great domain specific methodologies.
- Software and Hardware treat data differently
 - VSIPL++ represents data in discrete *blocks*
 - Hardware sees the data as a continuous *stream*
- Poses a problem of data exchange
- We developed an infrastructure to exchange data between VSIPL++ block data and the FPGA streaming data

Co-Design Solution



- Devise a memory adapter, a hardware unit on the FPGA, that directly accesses FPGA memory
 - Uses DMA to place/retrieve data into/from a flow controlled data stream.
- Create a user defined VSIP++ block (fpgaDense) based on the existing (Dense) block to facilitate the transfer of data to/from the FPGA memory

Memory Adapters



Control Data Flow

- The VSIIPL++ FPGA Dense block is LAZY!
 - Data is only transferred when necessary. Vectors created on the host are copied to FPGA memory only when FPGA function is initiated.
 - Conversely data is written to host memory, only when the host requests the data

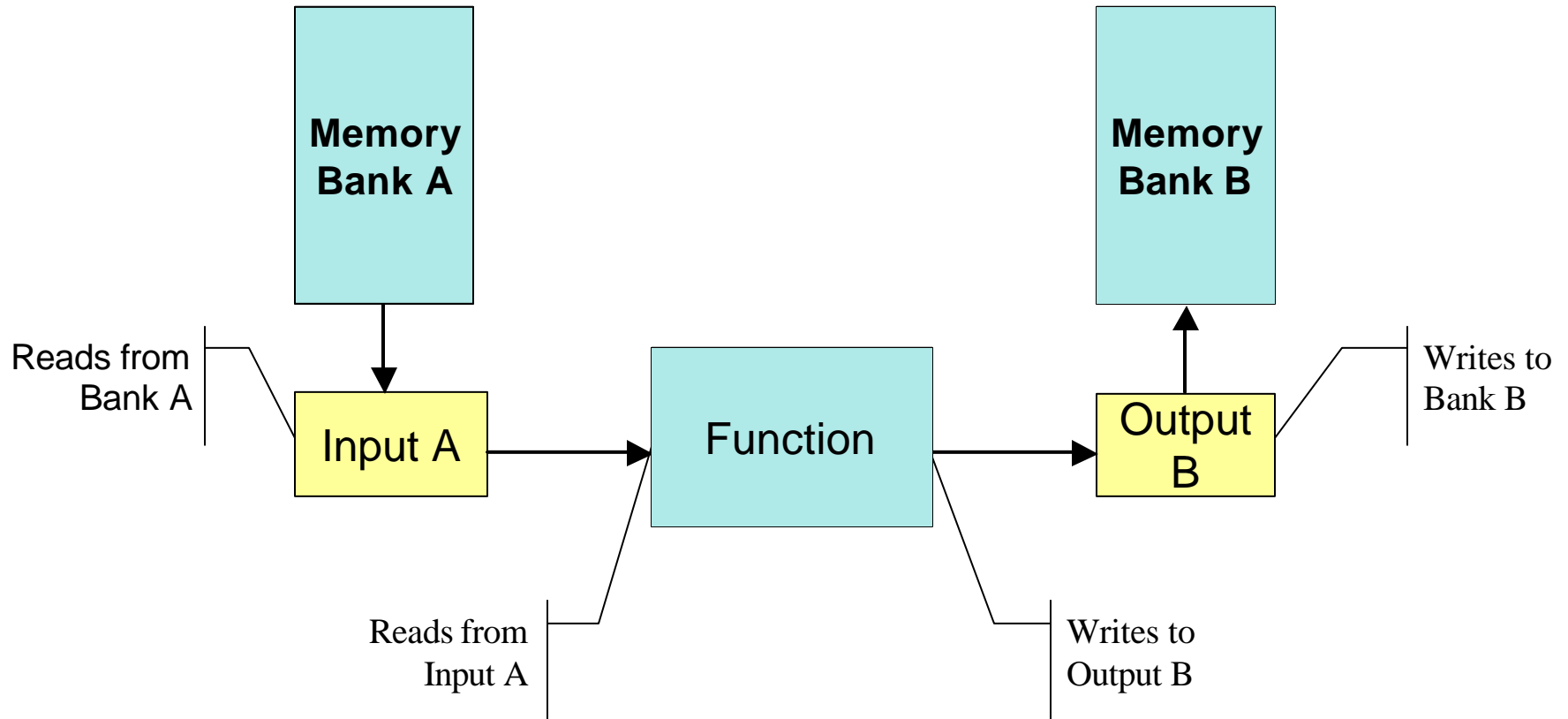
Software Representation



- The hardware is “Object Oriented” where each component object is described by connections.
- Therefore, we created software objects that directly corresponded to the hardware, also described by connections.

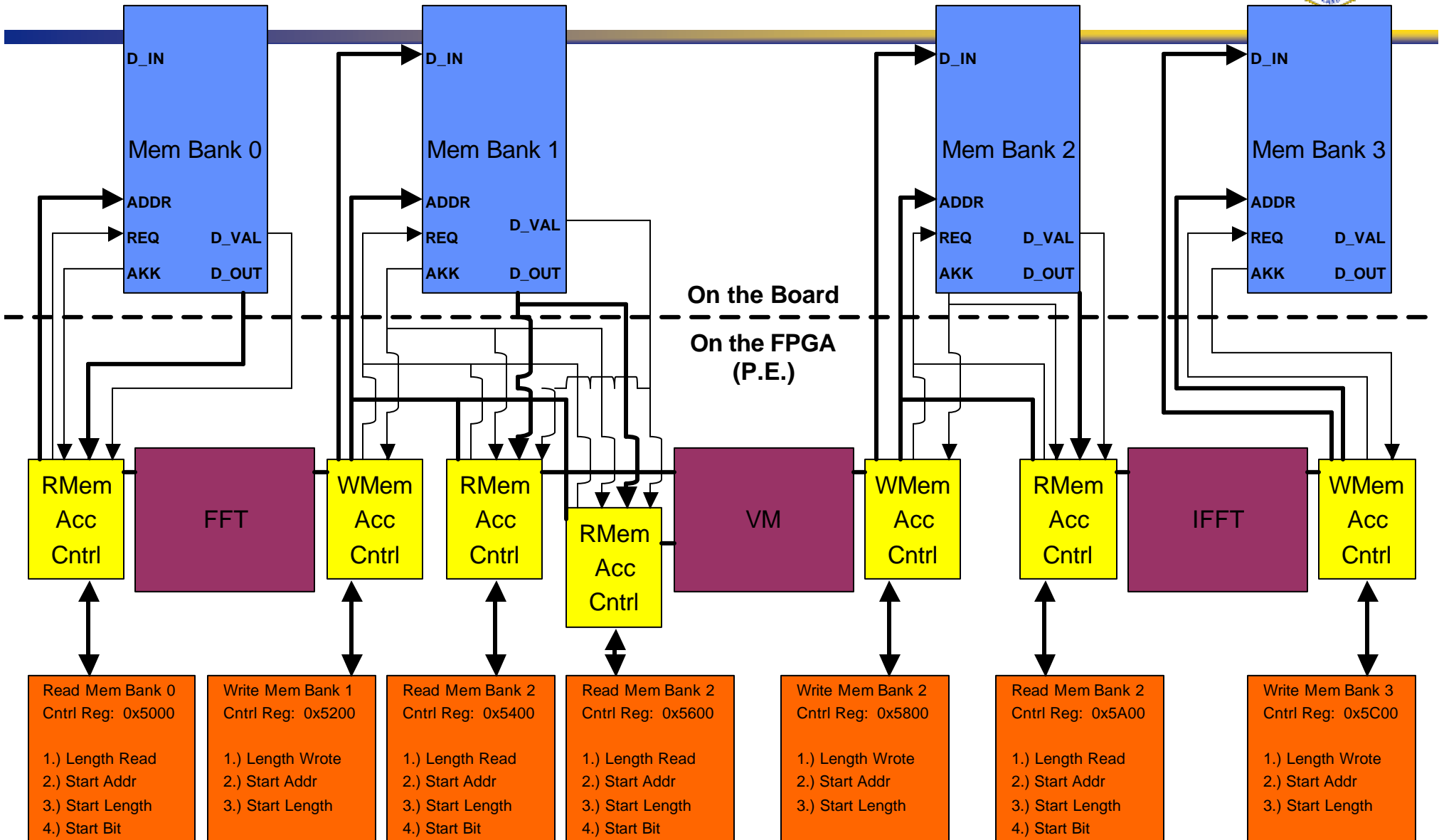
Facilitating the co-design of hardware and software.

Hardware Model

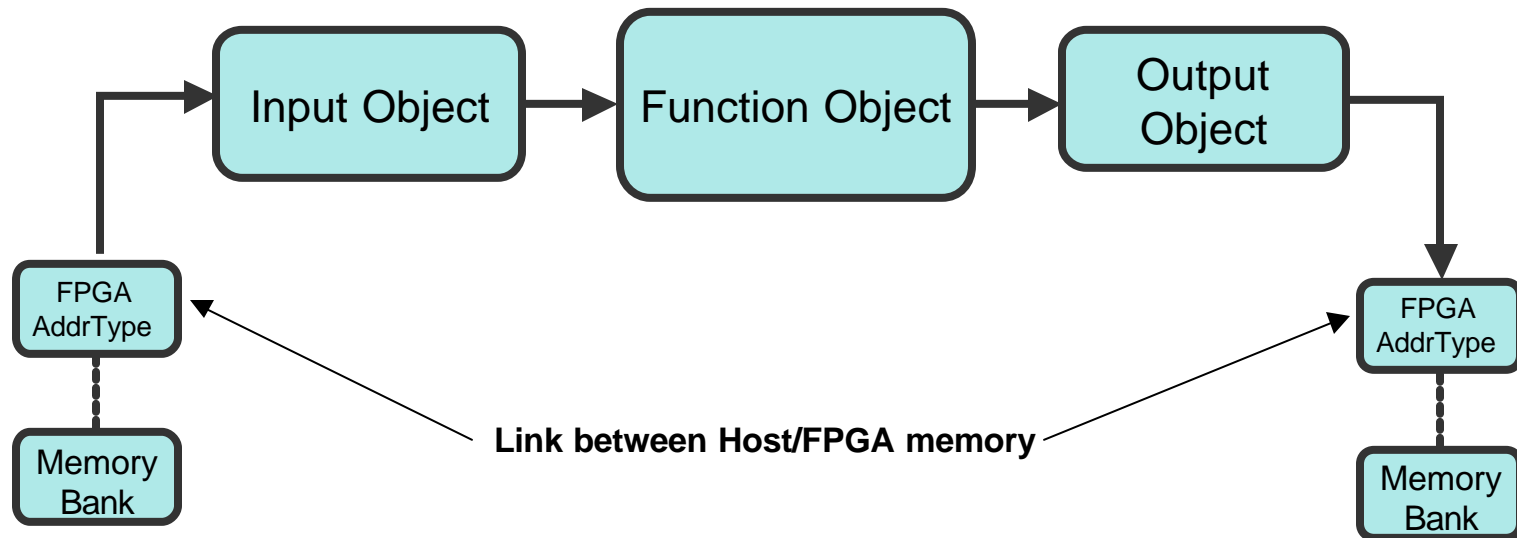


Arrows Represent Data Flow

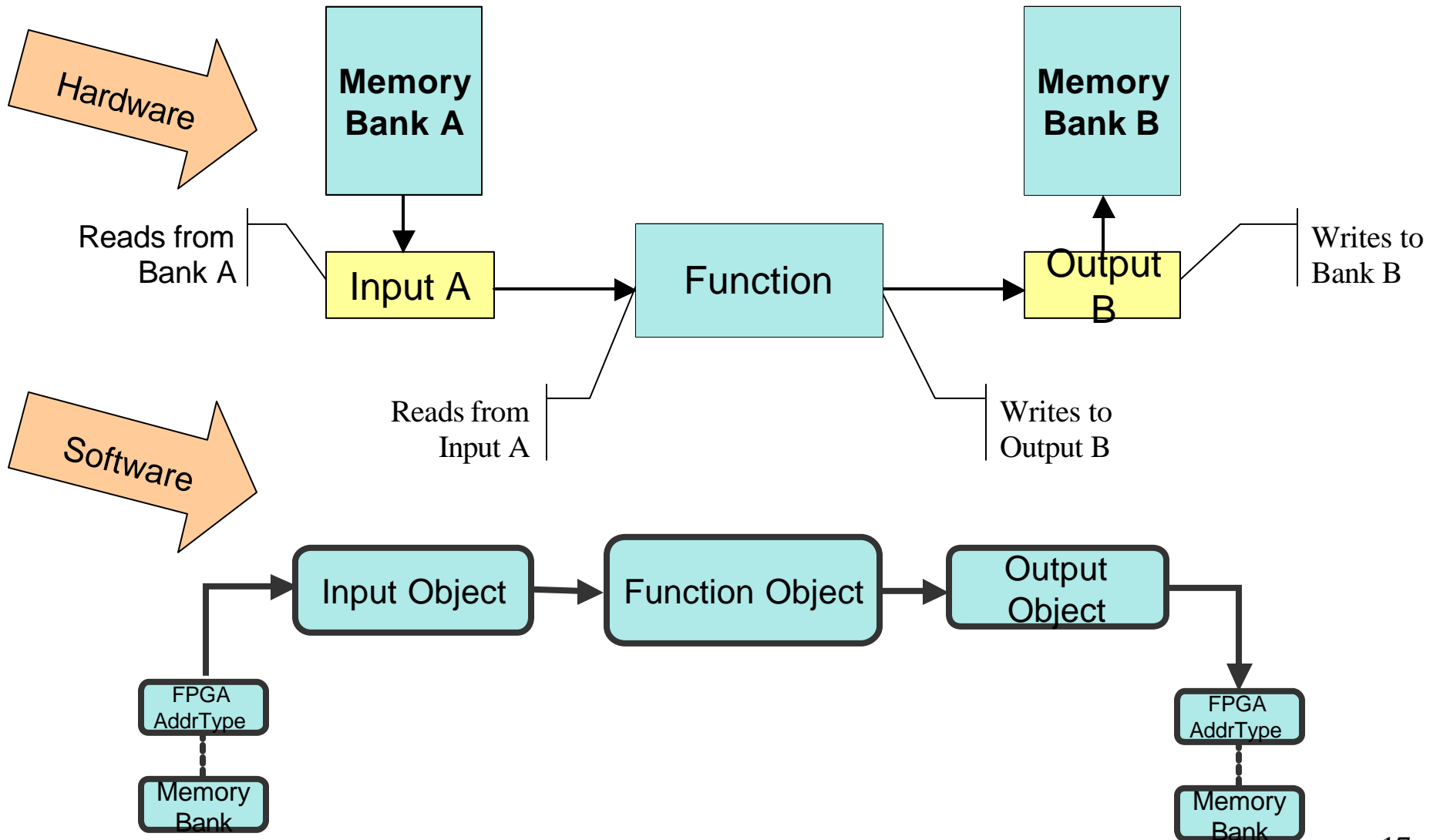
Hardware Block Diagram



Software Model



The Models



About the Models



Both models, each object is described connections

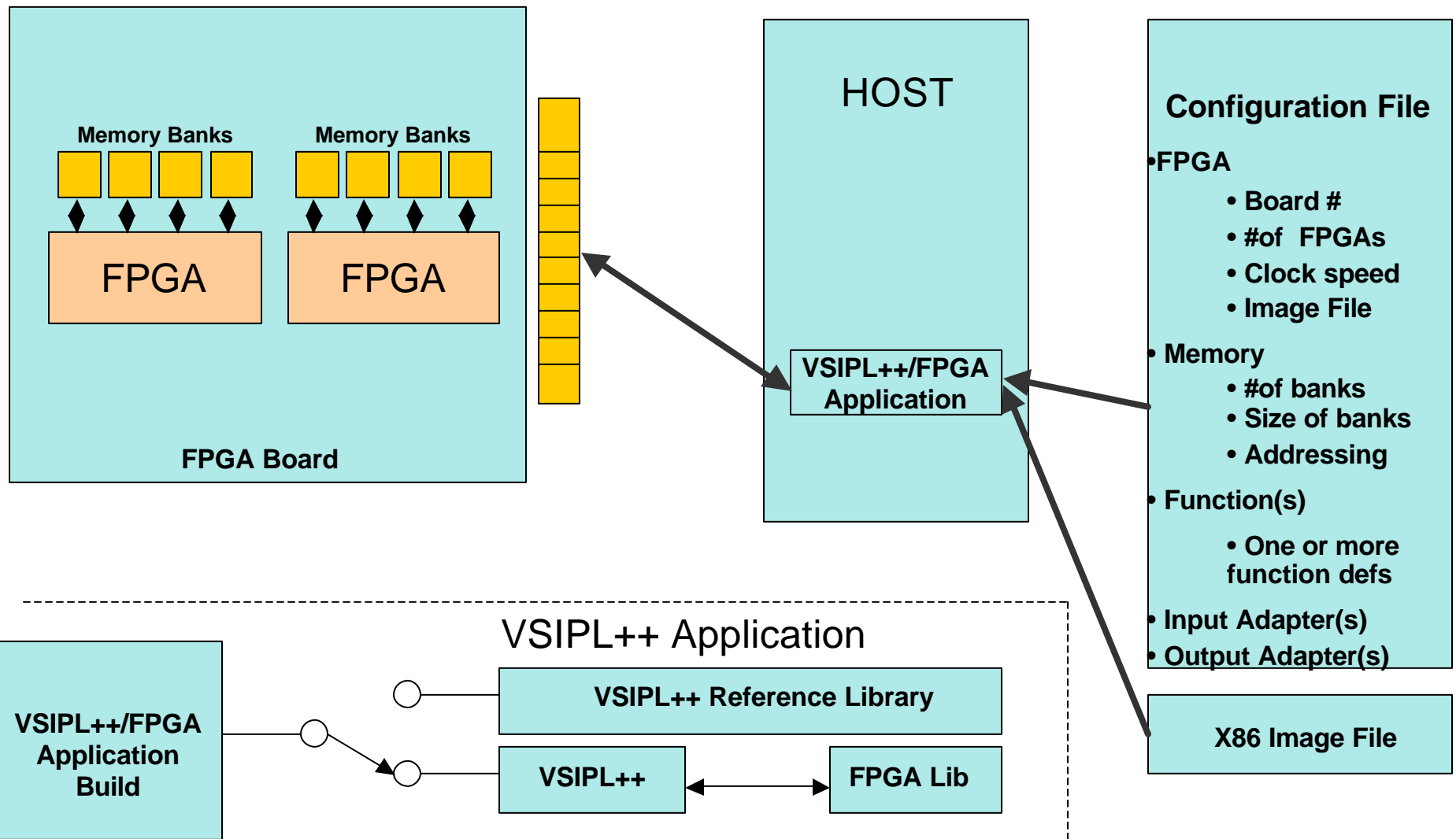
- Functions are described by I/O connections (data flow)
- I/O is described by memory connections
 - Or another I/O (an output can feed an input)
- Memory is described by the board and the processing element it is connected to.

Integrating VSIPL++ & FPGA



- A description of each hardware object is contained in a Configuration file.
- The system software reads the configuration file and creates a software object for each hardware object.
- The application simply calls the “function”, the system will “apply” the function.
 - Move the data (if necessary) & initiate the operation.

VSIPL++ / FPGA Interface



Configuration File



#Configuration File

PE = 0, 20 # Processing Element, processing speed mHz
CoreFileName=pe0.x86 # Filename of Binary code to program fpga with

#Memory Bank Definition

#Name, PE#, Size in bytes, DMA Write Port, DMA Read Port, radix assumed to be hex)

MemBank=Bank0, 0, 400000, 1000, 1200 # 4194304 bytes

MemBank=Bank1, 0, 400000, 2000, 2200

MemBank=Bank2, 0, 400000, 3000, 3200

MemBank=Bank3, 0, 400000, 4000, 4200

#Function Definition

FN=PC, input=R0, input=R1, input=R2, input=R3, Output=W0, Output=W1, Output=W2, size=256

FN=FFT0, input=R0, Output=W0, Size=256

FN=VMUL, input=R2, input=R1, output=W1, size=256

FN=IFFT, input=R3, Output=W2, Size=256



SAMPLE

Config File Cont.



#Input Adapter Defintion

#Name, Control Register Address, Name of assoc. memory bank, radix assumed to be hex

INPUT=R0, 5000, Bank0

INPUT=R1, 5400, Bank1

INPUT=R2, 5600, Bank1, persistant

INPUT=R3, 5A00, Bank2

#Output Adapter Definition

#Name, Control Register Address, Name of assoc. memory bank, radix assumed to be hex

Output=W0, 5200, Bank1

Output=W1, 5800, Bank2

Output=W2, 5C00, Bank3

Integration & Test



- Unit & System Level
 - As individual hardware units are developed, they can be immediately integrated into the software for testing, even before all components are complete.
 - Components can be integrated individually or grouped as sub-systems.
- Virtual Breadboarding
 - For larger systems, functions can be spread across multiple FPGAs.
- Hardware in the loop
 - Tightly coupled hardware/software can be easily constructed.

Applications



- Currently employing this technology in two applications:
 - Spaced Based Radar Embedded System design
 - With this method, a virtual bread board of the MTI/SAR system can be developed and tested on AFRL's hybrid cluster long before the actual hardware becomes available. (Not scheduled to fly until 2008).
 - *Rstream* signal processing library for application acceleration.
 - Develop a set of common Signal Processing functions which will be used in Space Based Radar.
 - Goal is to provide a library that can be used with VSIPL++

Status



- Prototype implementation completed
 - Annapolis Firebird Card using Xilinx VirtexE FPGAs
- Preliminary experiments on XILINX Virtex-II Pro which integrates a hybrid system, PowerPc processor

Future Work



- It is anticipated, that opportunities will arise as we gain experience with the streaming protocol
 - Possibly additional streaming objects (fifo's that logically bypass the fpga memory?)
- Computer Aided Design
 - Auto generate the configuration file
 - Auto merge of units that will be directly connected, optimizing redundant interfaces.
- Investigate areas outside of signal processing

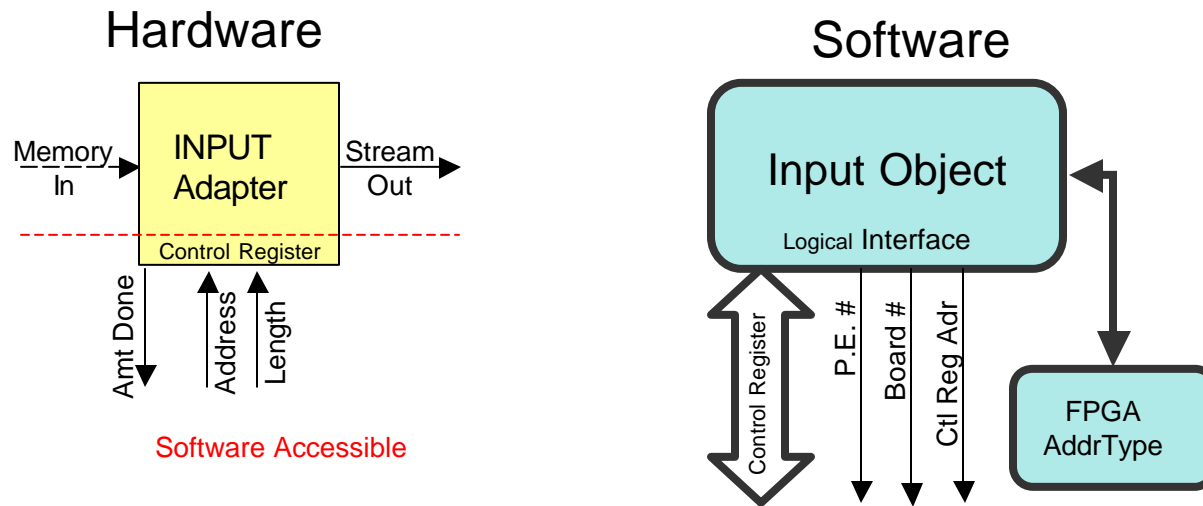
Conclusions



- The integration of VSIPL++ for software design with compose-able hardware design, provides a powerful design methodology for building hybrid hardware/software systems.
- VSIPL++'s performance, portability and productivity provide a growth path for parallel performance and hardware acceleration.
- The Stream hardware interface provides a simple mechanism by which hardware units can be composed together forming larger more complex units

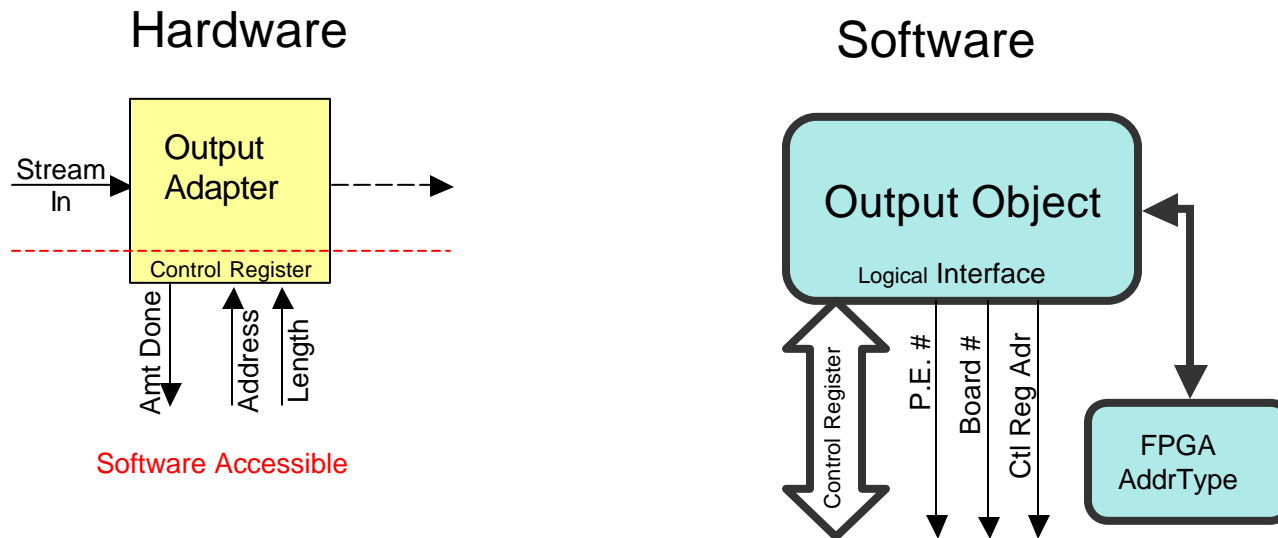
Addendum

Implementation Slides Follow



Input Responsibilities

- Control Data Flow
- Initiate Host memory to FPGA memory data transfers
- Initiate the data flow from FPGA memory to the function block



Output Responsibilities

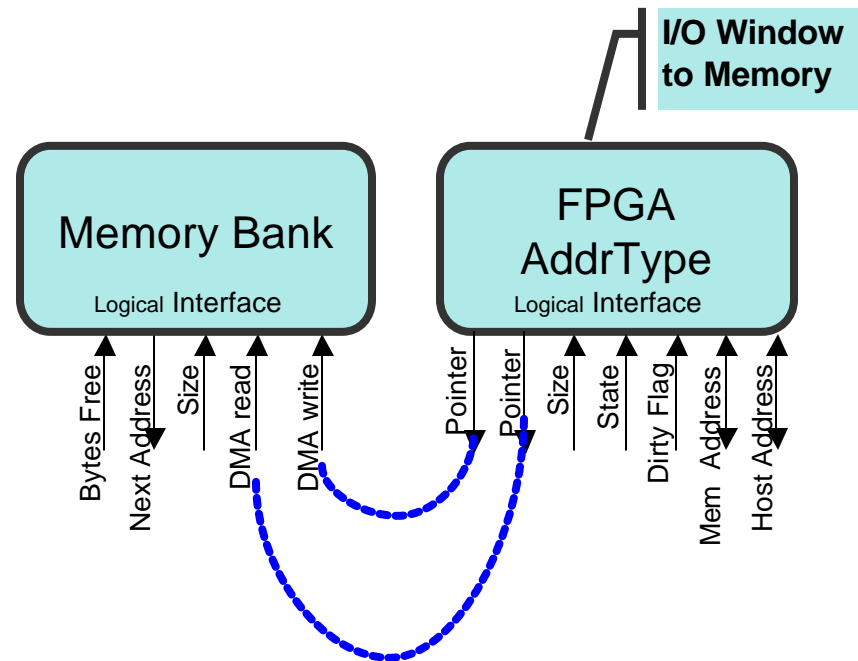
- Control data flow
- Initiate FPGA memory to Host memory data transfers
- Initiate the data flow to FPGA memory from the function block

Memory Bank



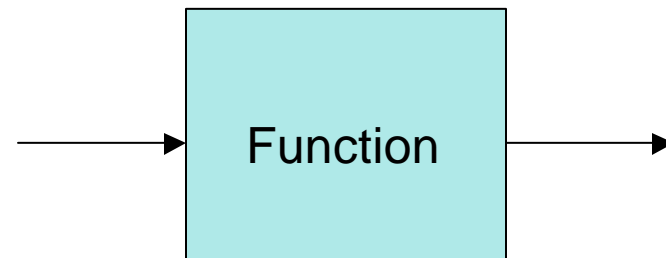
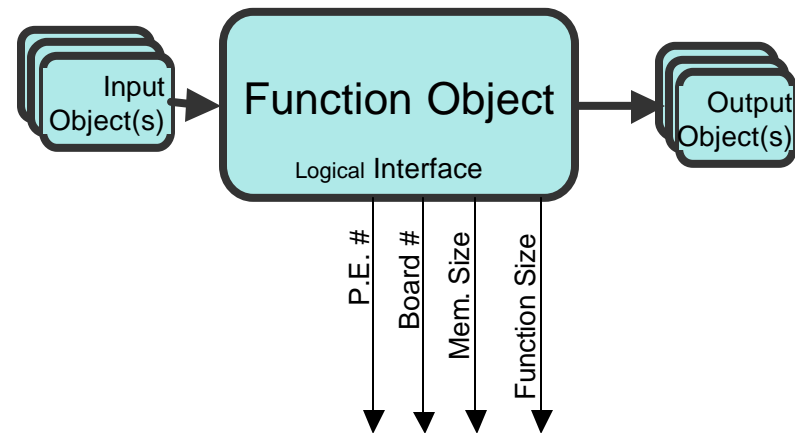
Responsibilities

- Manage 1 FPGA memory bank
- Maintain Total size
- Maintains Available Free memory
- Honors requests for memory
 - allocate
 - free
- FPGA AddrType Object is I/O adapters “window” to the Memory Bank
 - Host/FPGA memory transfers

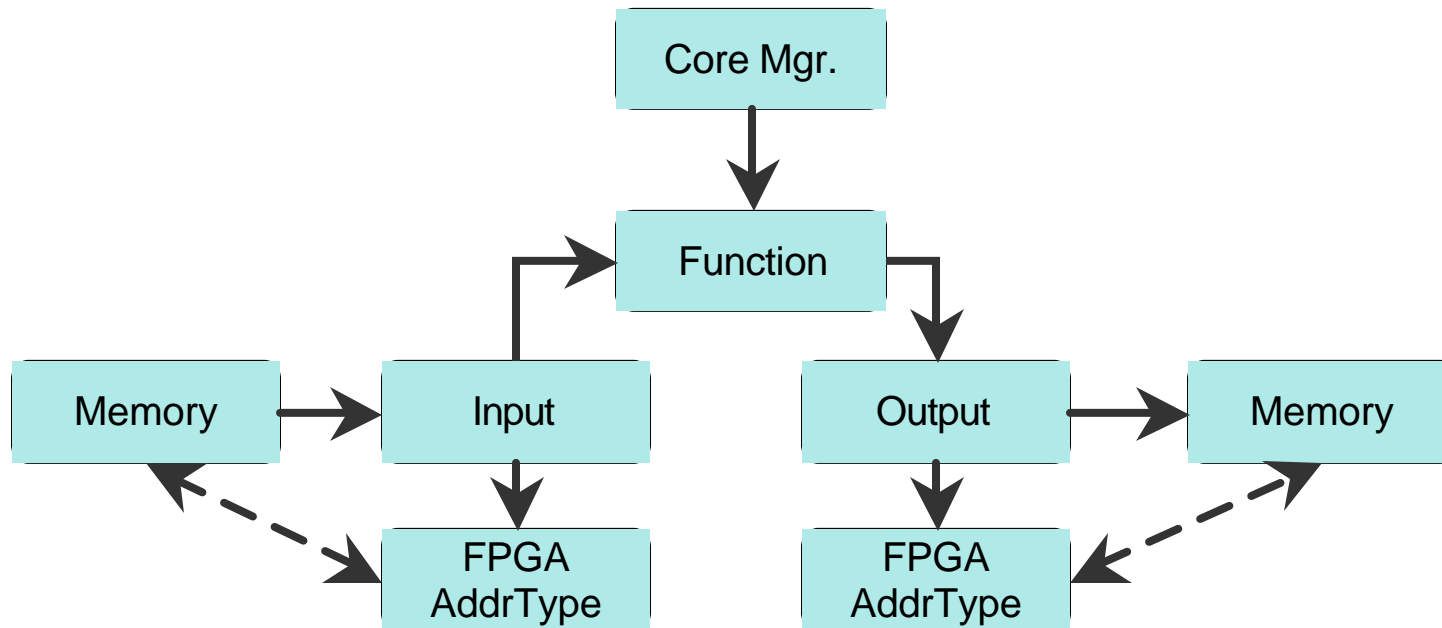


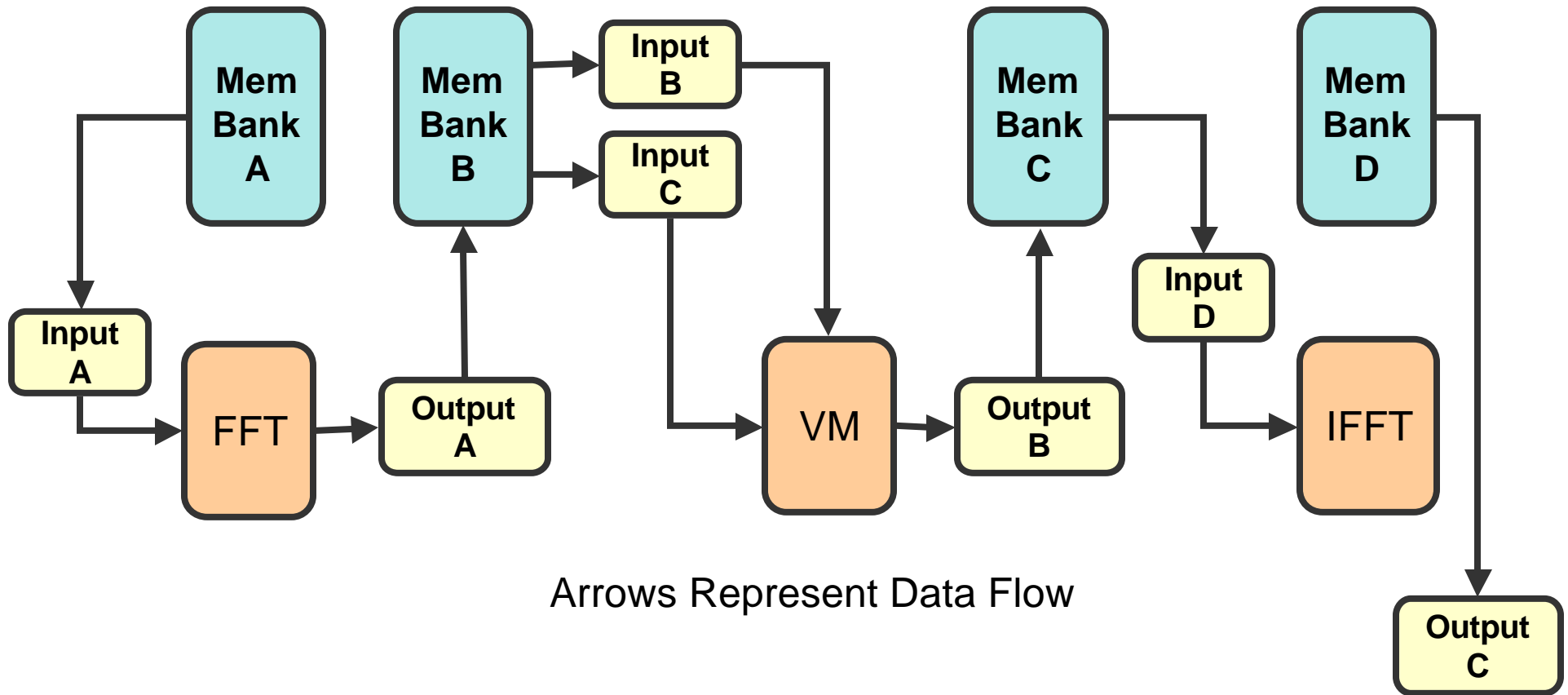
Responsibilities

- Instantiate Input Objects
- Instantiate Output Objects
- Activate/Halt Input Data Stream
- Activate/Halt Output Data Stream



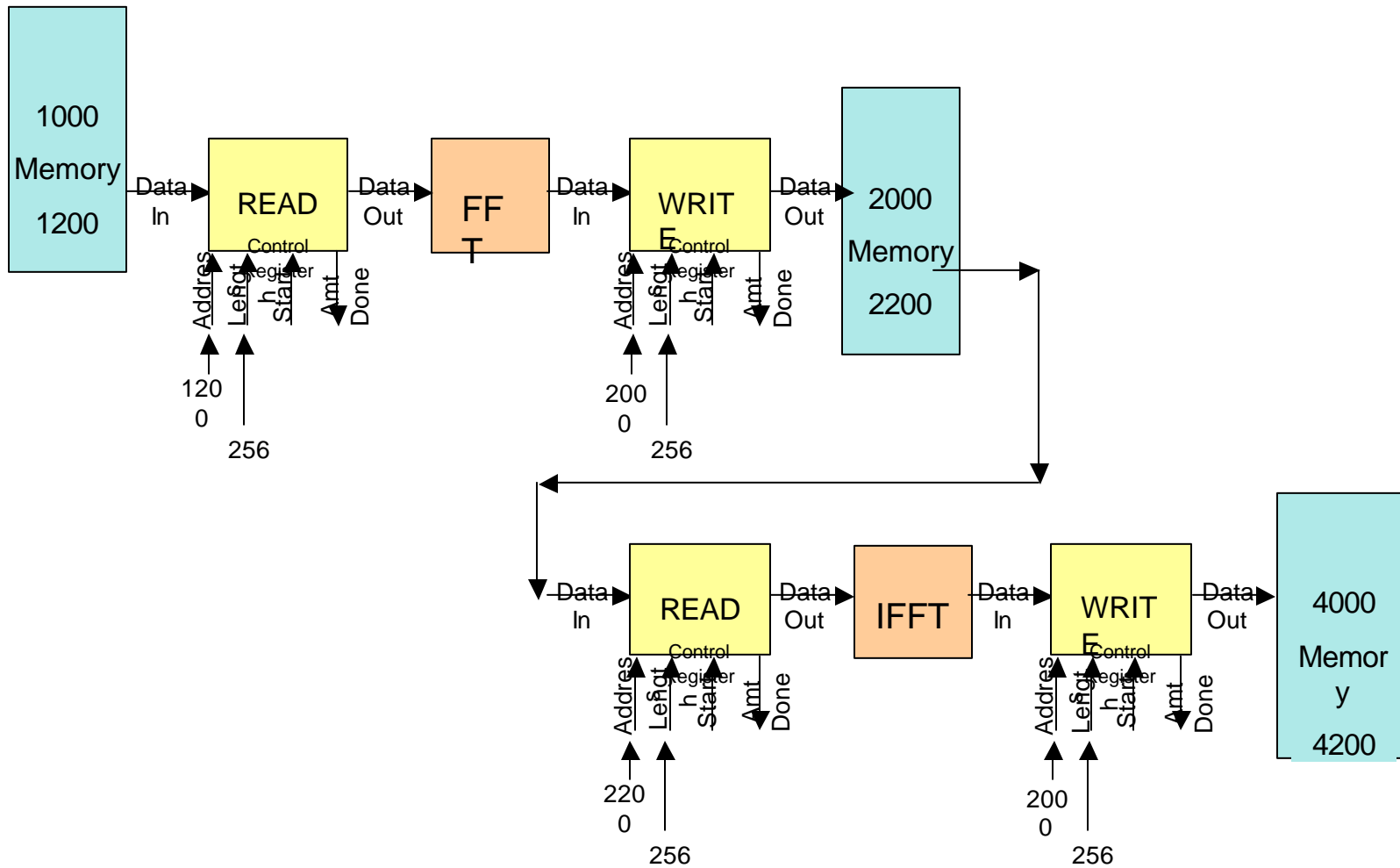
Object Relationships





PULSE COMPRESS

Hardware Design



PULSE COMPRESS