Configuring the Target Machine

Target Machine Description

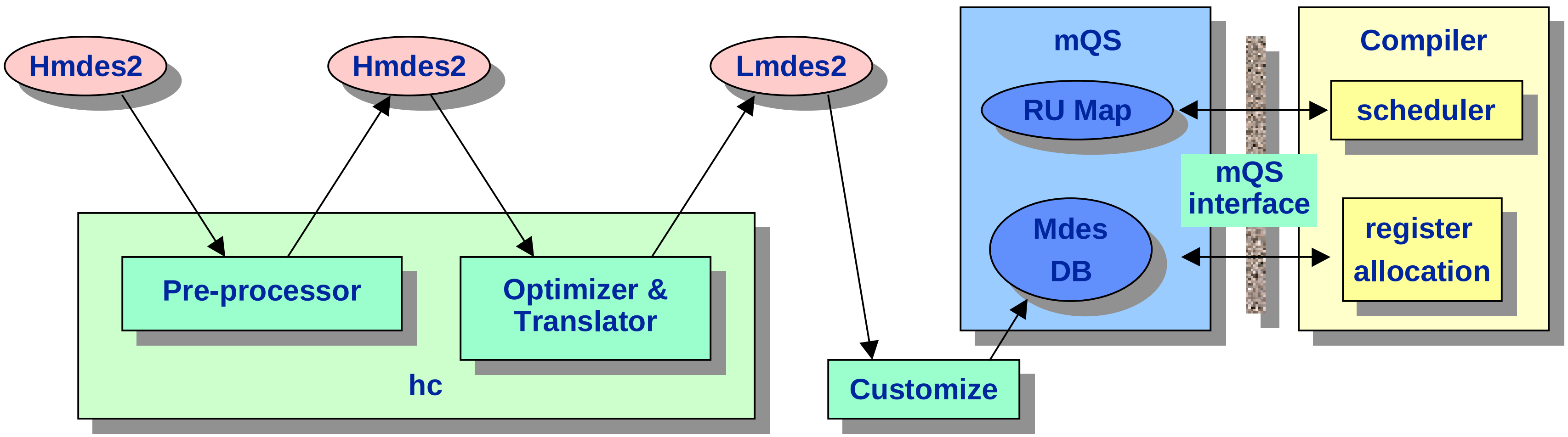
- Trimaran includes an advanced Machine Description facility, called Mdes, for describing a wide range of ILP architectures.
- It consists of
  - A high-level language, Hmdes2, for specifying machine features precisely
    - functional units, register files, instruction set, instruction latencies, etc.
    - Human writable and readable
  - A translator converting Hmdes2 to Lmdes2, an optimized low-level machine representation.
  - A query system, mQS, used to configure the compiler and simulator based on the specified machine features.

Target Machine Configuration

- Generally, it is expected that Trimaran users will make modest changes to the target machine configurations
  - within the HPL-PD architecture space
  - using the Trimaran graphical user interface (GUI) to modify an existing machine description rather than write a new one from scratch.
  - Very easy to change
    - number and types of functional units
    - number of types of register files
    - instruction latencies

Mdes Overview

- The goal: to minimize the number of assumptions built into the compiler back-end regarding the target machine



Machine Information Supplied to the Compiler

- For ILP code selection (determination of operand constraints with respect to the set of legal register files that may house the operands):
  - I/O descriptor: source / destination register file constraints for operations
  - Register file: the set of compatible register types
- Register Overlap
  - i.e., that have at least one bit in common
- Operand Latencies
  - source sampling / result update times for operations

Machine Information Supplied to the Compiler (cont)

- Legality of scheduling at a given time with respect to resource conflicts
  - Reservation table: resource usage over time for each operation

Relative Time	InstField02	InputBUS01	InputBus02	FPDivide_0	FPDivide_1	ResutBus01
0	X	X	X	X		
1				X		
2				X	X	
3					X	
4					X	X

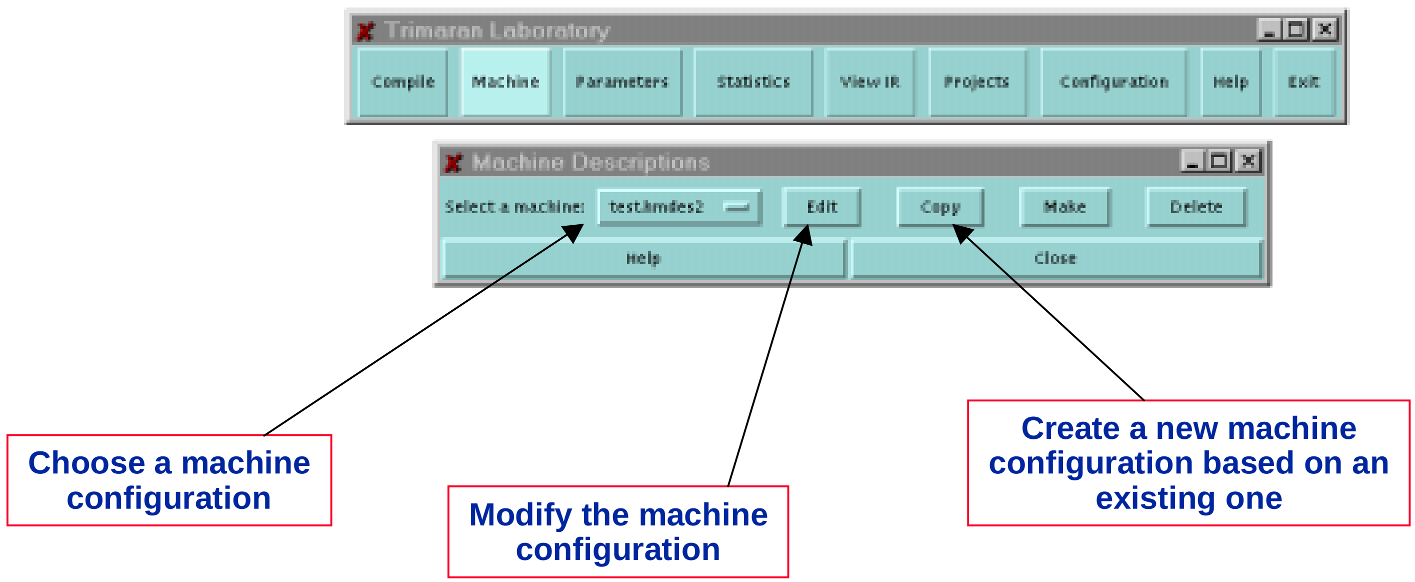
- Lifetime calculation
  - Latency descriptor: register allocation and deallocation times
- Register allocation options
  - Register file: the set of legal registers for allocation

Machine Descriptions

- There are two issues that a researcher must consider:
  – How can the features of a target machine be modified so that the changes are reflected in the code generated by the compiler and in the machine being simulated during execution?
    - Most common features can be changed via the GUI
    - Extensive modifications can be specified via an Hmdes2 description.
  – How can a new compiler module, implemented by a researcher, determine the features of the target machine?
    - The mdes Query System, mQS

Using the Trimaran GUI to configure an HPL-PD machine

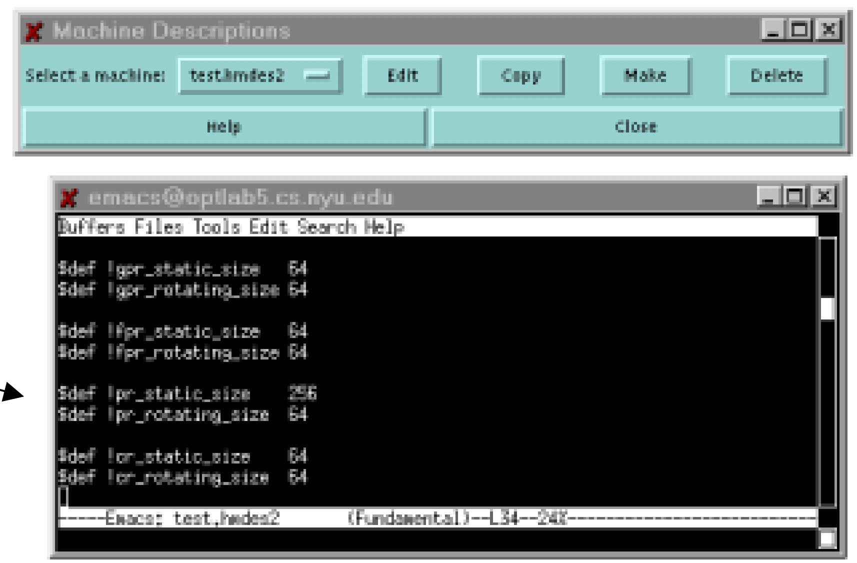
- The Trimaran system is delivered with a full Mdes description of several machines in the HPL-PD architecture space.
  - Machine features within the HPL-PD space are easily modified using the Trimaran GUI.
    - Functional units, register files, instruction latencies



Using the Trimaran GUI to configure an HPL-PD machine (cont)

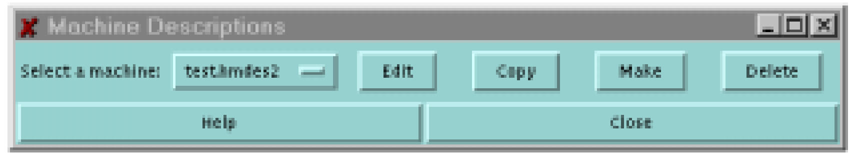
- When the Edit button is clicked, an Emacs window opens a configuration file for editing.
  - This file is read by the Hmdes2 preprocessor and translator to create a new Lmdes2 machine description.
- Changes to the configuration file are reflected in the target machine when the Make button is clicked.

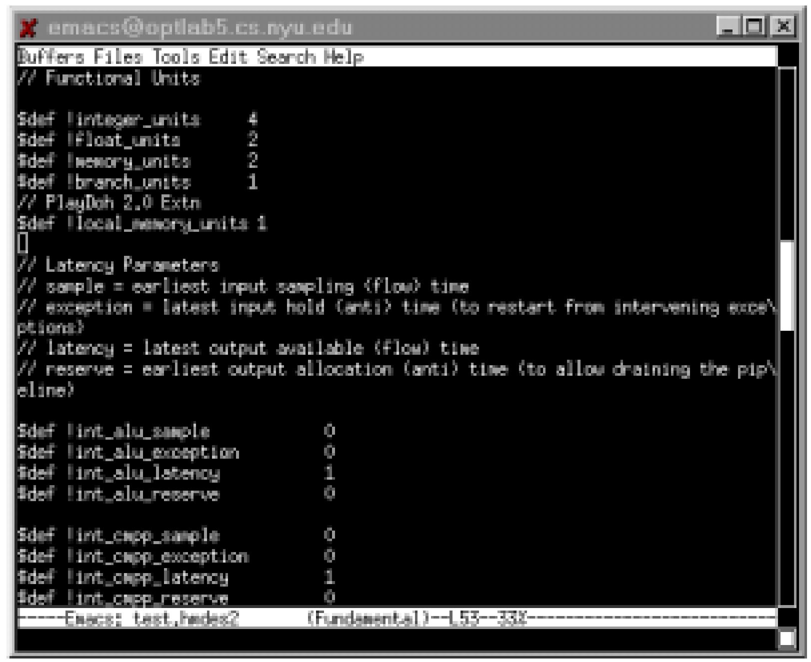
This portion of the file specifies the number of static and rotating registers in the various register files.



Using the Trimaran GUI to configure an HPL-PD machine (cont)

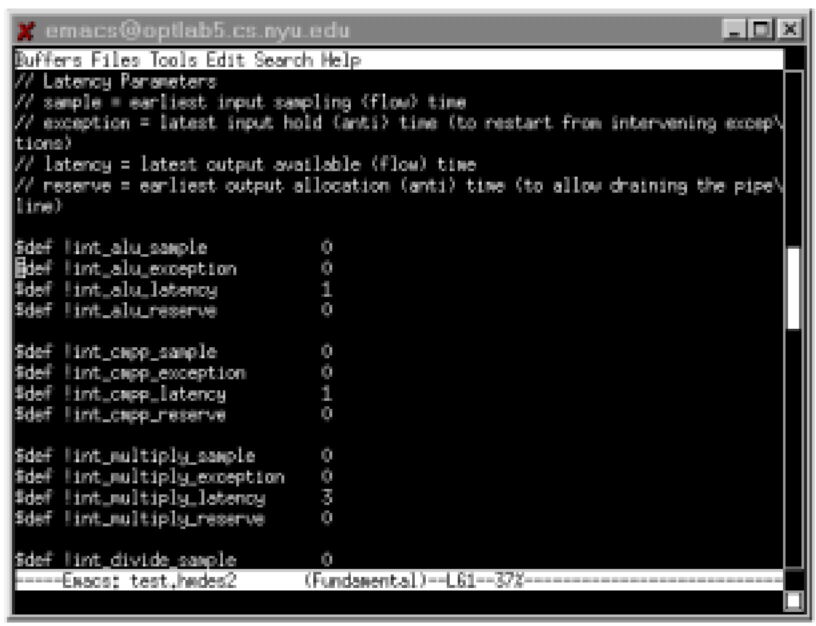
- With a few keystrokes, the machine can be radically changed.
  - From an essentially sequential machine (very few functional units)
  - To a highly parallel machine.





Using the Trimaran GUI to configure an HPL-PD machine (cont)

- The machine configuration changes via the GUI can be quite detailed.
  - In this case, the precise latencies of operations can be modified.
    - When the input registers are sampled.
    - When the value in the output register is available.
    - Etc.



Describing a Machine Using Hmdes2

- If more extensive changes to a machine need to be made than can be handled in the GUI, the user can describe the machine using Hmdes2.
  - High-level machine description language
- There is a limit, however, to the extent that a machine can be modified and still be the target for the Trimaran compiler, and be simulated using the Trimaran simulator.
  - The machine must remain in the HPL-PD architecture space.
  - The instruction set cannot be significantly changed.
- The GUI is the recommended method for modifying the target machine.
  - However, Hmdes2 is a very interesting mechanism…

Hmdes2

- Hmdes2 is a schema expressed in DBL
- DBL: an incremental relational database description language

$Section_1$	$field_1$	$field_2$	...
$record_1$			
$record_2$			
...			

$Section_2$	$field_1$	$field_2$	...
$record_1$			
$record_2$			
...			

...

- Text Macroprocessor
  - File inclusion
  - Macro-variables, shell environment variables
  - Recursive variable replacement (textual)
  - Fixed/floating numeric expression evaluation
  - If-then-else
  - For-loop (counted and list ranges)

Register

- Schema

```
CREATE SECTION Register
  OPTIONAL overlaps(LINK(Register)*);
{}
```

- Example

```
SECTION Register {
  GPR0(); GPR1(); … GPR63();
  'GPR[0]'(); 'GPR[1]'(); … 'GPR[63]'();
  ...
  CR0(overlaps(PR0 … PR31));
  ...
}
```

Register File

- Schema

```
CREATE SECTION Register_File
  REQUIRED width(INT);
  OPTIONAL virtual(STRING*);      // generic register types supported
  OPTIONAL static(LINK(Register)*);
  OPTIONAL rotating(LINK(Register)*);
  OPTIONAL speculative(INT);      // 0=non-spec. 1=spec.
  OPTIONAL intlist(INT*);         // literal values
  OPTIONAL intrange(INT*);
  OPTIONAL doublelist(DOUBLE*);
{}
```

- Example

```
SECTION Register_File {
  RF_i(width(32) virtual(i) speculative(1)
     static(GPR0 … GPR63) rotating('GPR[0]' … 'GPR[63]'));
  LF_s(width(6) virtual(l) intrange(-32 31));
  ...
  LF_l(width(32) virtual(l));     // generic literal file (for Elcor)
  RF_u(width(0) virtual(u));      // generic bit-bucket (for Elcor)
}
```

Reservation Table

- Schema

  CREATE SECTION Reservation_Table

  REQUIRED use(LINK(Resource_Usage)*);

  {}

Relative Time	InstField02	InputBus01	InputBus02	FPDivide_0	FPDivide_1	ResultBus01
0	x	x	x	x		
1				x		
2				x	x	
3					x	
4					x	x

- Example

```
SECTION Reservation_Table {
  RT_null(use());        // null reservation for dummy ops
  RT_i0(use(RU_i0));
  RT_i1(use(RU_i1));
  ...
}
```

Operation Latency

- Schema

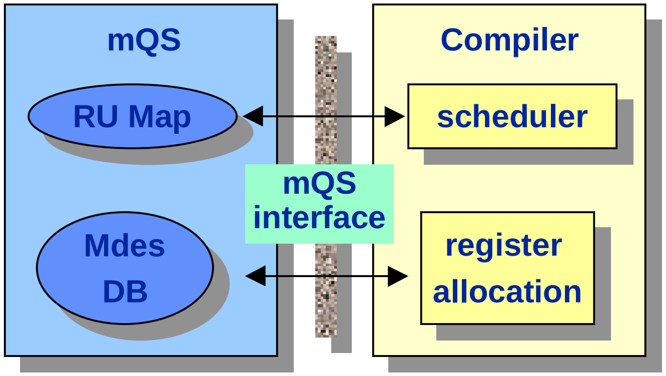
```
CREATE SECTION Operation_Latency
  OPTIONAL dest(LINK(Operand_Latency)*);        // Tr
  OPTIONAL src(LINK(Operand_Latency)*);         // Ts
  OPTIONAL pred(LINK(Operand_Latency)*);        // Ts
  OPTIONAL exc(LINK(Operand_Latency));          // Tx (one for all inputs)
  OPTIONAL rsv(LINK(Operand_Latency)*);         // Ta
  OPTIONAL sync_dest(LINK(Operand_Latency)*);   // Tr (for sync ports)
  OPTIONAL sync_src(LINK(Operand_Latency)*);    // Ts (for sync ports)
{}
```

- Example

```
SECTION Operation_Latency {
  OL_int(dest(time_int_alu_latency … time_int_alu_latency)
       src(time_int_alu_sample … time_int_alu_sample)
       pred(time_int_alu_sample)
       exc(time_int_alu_exception)
       rsv(time_int_alu_reserve … time_int_alu_reserve)
       sync_dest(time_int_alu_sample time_int_alu_sample)
       sync_src(time_int_alu_sample time_int_alu_sample));
}
```

The Compiler/Machine Description Interface

- The interface between the compiler and the machine description is the mdes Query System, mQS.
  - New modules implemented by researchers will need to use the mQS.
- The compiler queries mQS via a set of C++ procedures.
  - Each class of machine feature corresponds to a separate C++ procedure.



mQS Interface Examples

- Register file parameters

```
void MDES_reg_names(List<char*>& regnames);  // list all register files
int MDES_reg_static_size(char* regname);
int MDES_reg_rotating_size(char* regname);
int MDES_reg_width(char* regname);
```

- Operation parameters

```
int MDES_src_num(char* opcode) ;        // excludes predicate input
int MDES_dest_num(char* opcode) ;
Bool MDES_predicated(char* opcode) ;
Bool MDES_has_speculative_version(char* opcode);
```

- Latency parameters

```
void MDES_init_op_io(char* opcode, char* iodesc);
int MDES_flow_time_io(IO_Portkind portkind, int portnum); // Tr, Ts
int MDES_anti_time_io(IO_Portkind portkind, int portnum); // Tx, Ta
void MDES_branch_latency(char* opcode);                  // branch Tr
```

Resource Manager Functions

- Resource table manipulation

```
void RU_alloc_map(int maxlength);
void RU_delete_map(void);
void RU_print_map(FILE *mout);
void RU_init_map(Bool modulo, int length);
```

Relative Time	IntALU_0	IIntMult_1	InputBus_0	InputBus_1	ResultBus_0	ResultBus_1
0	w	x	x	w		
1		z	z	z	w	
2	y		y	y		
3					y	x
4						z

- Operation scheduling

```
void RU_init_iterator            // initialize scheduling request
  (char* opcode, void* op, char* iodesc, int time);
Bool RU_get_next_nonconfl_alt
  (char** opcode, int* priority); // return alternative, if successful
void RU_place(void);              // commit alternative
void RU_get_conflicting_ops(Hash_set<void*>& ops);
void RU_remove(void* op, char* iodesc, int time);
void *RU_at(int time, int res_index);
```

Summary

- Reconfiguring the target machine is quite easy
  - GUI speeds up the process substantially for modest changes.
  - Extensive changes can be made using Hmdes2
    - there are plenty of sample Hmdes2 files to look at.
- Adding new machine-dependent compilation modules is also quite easy
  - mQS provides a clean interface between the compiler and the machine description.