Instruction Unit Group:
Program Sequencer

Vlad Branzoi
Jerry Kao
Overview

- Program Flow
  - Next Instruction Address Generation
  - Loop Handling
  - Jump/Call Instructions
  - Conditional Execution

- Interrupt Handling
  - Stack Management

- Unit Status
  - Status Registers (ALU/MAC operations status)
Signals

27 Input signals, ~3 output signals

- funct_field<0:3>
- cond_code<0:3>
- addr_jumpcall<0:13>
- last_loop_addr<0:13> + term_cond<0:3>
- flag_bits<0:7>
- mode_bits<0:13>
- stack control lines (push/pop)
- register writes
- mux and buffer selects

- DMD_bus<0:15>
  - stack dumps
  - register reads/moves
- PMA_bus<0:14>
  - next_addr
- condition return to decoder
Additional Signals

- Status signals from ALU and MAC
  - AZ – ALU result zero
  - AN – ALU result negative
  - AV – ALU Overflow
  - AC – Carry
  - AS – ALU X Input Sign
  - MV – MAC Overflow
  - SS – Shifter Input Sign
  - more detail in ASTAT section
Block Diagram

Counter Block
Block Diagram

Counter Block

Status Registers

Loop Block
Custom Additions

- Function Field Codes
- Status bit generation
- Mode bits handling
- Flag In/Out control
- Stack control
Loop Control
Loop Control (continued)

How are Loops handled?

DO LABEL UNTIL Termination Condition

Instruction 1
Instruction 2
...

Label: Last Instruction

Labels are 14 bit addresses to the last instruction in the loop and the Termination Condition is one of the following:

- EQ
- NE
- LT
- GE
- LE
- GT
- AC
- NOT AC
- AV
- NOT AV
- MV
- NOT MV
- NEG
- NOT AC
- NOT AV
- NOT MV
- POS
- CE
- FOREVER
Conditional Instruction

- The condition logic uses the data in the Arithmetic Status Register to determine whether the conditions are true.
- It is used for Jump, Jump/Call, and Loop instruction, and also by Conditional Execution of certain instructions.
- The output of the condition instruction is not only sent to the next address generation, but also to the instruction decode for conditional execution of instructions.
- Timing issue
Counter Block
Counter Block (continued)

- **When do we use counter**
  - The counter is used as a condition
  - Mostly with loops but can be used by any conditional operation

- **Special considerations**
  - Counter Register must be loaded with value BEFORE it is used in a loop or condition
  - Special care must be taken when overwriting counter inside a loop that depends on it
Status Block
Status Block (continued)

- **Status Registers**
  - Arithmetic Status Register (ASTAT)
  - Stack Status Register (SSTAT)
  - Mode Status Register (MSTAT)
  - Interrupt Mask (IMASK)

- **Status Stack**
  - Used to store the Status Registers during an Interrupt
  - When an Interrupt occurs the ASTAT, MSTAT and IMASK are pushed onto the stack
  - Stack is deep enough to handle all interrupts (in our case 1)

- **Status Bits generation + Masks**
Status Register

Arithmetic Status Register (ASTAT)

- Input from ALU and MAC units
- Used by Conditional Logic to determine the validity of condition and termination codes
- Must be written with a mask in order to ensure that bits that do not require changing are left unchanged
Status Register

Stack Status Register (SSTAT)

- Keeps track of status of all the stacks (whether they are overflowed or empty)
- Used by Stack Control and any operations on stacks
- When an overflow occurs, the newest information is lost (older information has precedence)
Status Register

Mode Status Register (MSTAT)

- A method to enable different operating modes of the DSP
- Bits are enabled by MODE CONTROL instruction
- Do we need it?
- Encoding of Mode Bits in the Instruction is unknown
Status Register

Interrupt Mask (IMASK)

- One bit mask, since only one interrupt required
- Used to disable the Interrupt
- A part of the status stack.
- It is saved when an interrupt is called
- It is set when the interrupt is called so that it cannot be interrupted by the same interrupt
Next Instruction Address Generation
Next Instruction Address Generation

The next instruction address is determined from 5 sources:

- PC Counter + Incrementer
- PC Stack (Loop/Call Handling)
- Jump/Call Address embedded in the instruction
- Fixed Interrupt Address
- PMA Bus from DAG2 (Jump/Call w/ I Register)

The Next Address Source Select (NASS) does the bulk of the selection logic.
Next Instruction Address Generation

Case 1: Normal Incremental Execution

The default program flow:

- Send the value of PC+1 to the PMA bus
- Load this address back into the Program Counter
Next Instruction Address Generation
Case 2: Loop handling and RTI

- The top value of the PC Stack is sent to the PMA_bus and to the Program Counter as the next address.
- In case of a LOOP the value is simply read while the LOOP continues.
- For Return Interrupt (RTI) the value is Popped from the PC Stack.
Next Instruction Address Generation
Case 3: Jump and Call instructions

- Address of Jump/Call is encoded in the instruction and is sent by the instruction decoder unit.
- In the case of Jump/Call with I Register, the address comes from the PMA bus from the DAG2 unit.
Next Instruction Address Generation

Case 4: Interrupt Handling
Interrupt handling is easy for our case since there is only 1 interrupt.

On an interrupt following events happen:

- PC Incrementer is disabled
- PC is pushed on the PC Stack
- A fixed Interrupt Address is sent to the PMA bus
- All status stacks are pushed
- Interrupts are disabled (while the interrupt is being serviced)
Issues

- **Unknown decodings**
  - We do not know the decoding for the Flag Out Bits instruction
  - We do not know the decoding for the Spp bits in the Stack Control instruction

- **Timing**
  - Out unit contains several long signal lines which can introduce timing and racing problems
  - A lot has to be accomplished in 1 clock cycle
THE END