ELG 6163
Digital Signal Processing Microprocessors, Software and Applications
Prof. Miodrag Bolic

Project
By: Peter Farkas
Date: April 12, 2007

Multiprocessor System on Chip Architecture for Kalman Filter based Speech Enhancement Algorithms

1. Introduction

The purpose of this project was to find a multiprocessor architecture that is well suited for implementing a subset of speech enhancement algorithms. Once found it was implemented on an FPGA using Nios II processors, in addition to other components including memory blocks. The implemented architecture and the algorithm running on it are described in the following sections.

2. Architecture

The designed architecture shown in Fig. 1 consists of five processors. The processors are in a master-slave configuration. The master processor can read in files from a read-only file system on the flash. It can distribute work to the slave processors by utilizing either the message passing, or shared memory methods. For message passing, five mailboxes have been incorporated into the system. These mailboxes are basically fifos, they allow messages to be sent and buffered. Each slave has a mailbox between them and the master. The master also has its own mailbox which all the slaves are connected to. The memory used for this mailbox has dual ports; this allows the master to access the mailbox with out the need for arbitration.

The mailboxes used for message passing work well for short messages, such as commands, but other facilities are available for transferring medium to large amounts of data. Connected to each slave is a tightly coupled memory, currently 2.5 KB large. This gives each slave their own private memory which they can access without arbitration and at a faster rate than on-chip memory. A larger shared memory with 256KB is also available. It has dual ports, allowing the master to access it without arbitration. However the slaves have to share a common port, this may negatively impact performance if more than one slave tries accessing it at the same time.

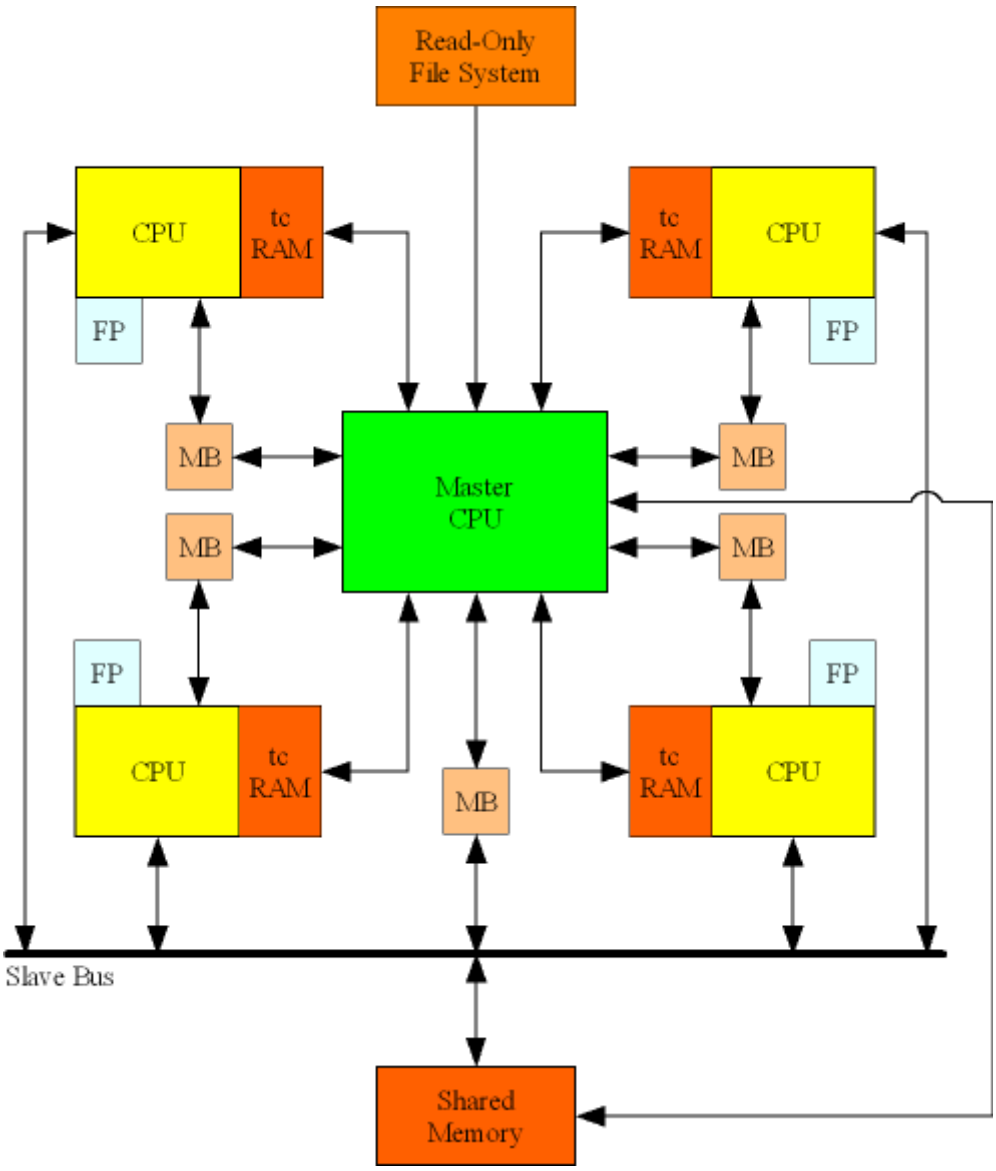

Fig. 1. Architecture

The master processor is a Nios II/s core, with 512 KB of instruction cache. The slaves are implemented using Nios II/f core, containing 512 KB of instruction cache, a port for tightly coupled data memory, and a floating point unit. The floating point units increase the processing time when adding or multiplying single precision floating point numbers. All the processors contain hardware multipliers, which are implemented by DSP blocks. None of the processors have any data cache, this is because the data that they will be accessing resides on-chip already.

Each processor is given 3 MB of memory on the off-chip SDRAM, the master however has 4 MB. This is where the program resides, as well as the data memory, stack and heap. 1 MB of off-chip SRAM is also available to the processors. In addition to these components there are two performance counters, a JTAG UART, and two standard UARTs. The performance counters can be used to profile the system, the master has access to one, and one of the slaves has access to the other. To see the status of the program the master can output messages to the console via the JTAG UART. The two standard UARTs are connected to two of the slaves, this is useful when debugging, or to print out performance results measured by the performance counter.

3. Algorithm

The algorithm and the original code was written by Jan Kybic, and it may be found at http://cmp.felk.cvut.cz/~kybic/dipl/. The code was modified to work on the Nios processors. All the commands for communicating between the processors were changed to allow for message passing and sharing via shared memory. The main loops of the code running on the master and slave processors are given below.

3.1 Master Processor

The main loop of the program running on the master processor is described. The loop is a for loop, which runs through all the frames of the audio file. For each frame the power spectrum is calculated, along with the smoothed estimates and the noise is estimated. For the first 64 frames, the startup phase, no filtering is done. The next four frames are distributed to the four slaves. After this the frames are distributed on a first come first served basis to the other processors. The code follows:

```
for i = 0:nwin                          //process the frames one by one

    ind = i*64                          // beginning of the window

    calculate_power_spectrum(ind)       // calculate the spectrum first
    smooth_power_spectra(p)             // calculate the smoothed estimates
    estimate_noise(p)

    if i < 64 then                      // startup phase - do not filter
      for j = 0:255
        output[j] = 0.0
      end
    else
      subtraction_rule(p)
      if i-64 < 4 then                  // initial task distribution
        give_work(i,i % 4)
      else
        j = process_message()           // process available message
                                        // returns the free slave j
        give_work(i,j)                  // give work to slave j
      end
    end
end

// process pending messages
while (pending > 0)
      process_message()
end
```

3.2 Slave Processor

The slaves block waiting for an incoming message from the master. The value received is the frame number which shall be processed. The values the master calculated from doing the power spectrum estimation, and the smoothed estimate were placed into the slaves tightly coupled memory. The slave sets a pointer to the beginning of the frame, which is located in shared memory. The slave estimates the AR parameters, and starts the Kalman filter. Once done a message is posted to the master's mailbox. The code follows:

```
while true
      // receive message from master
      r = altera_avalon_mailbox_pend()      // frame number to process
      ind = altera_avalon_mailbox_pend()

      // set pointers of sss and snps in tightly coupled memory
      sss = (double *)TC_RAM
      snps = (double *)(TC_RAM + 129)

      // set pointer to the beginning of the frame of the audio file
      input = (double *)SHARED_MEMORY_BASE + ind

      estimate_AR_parameters(sss)
      estimate_AR_parameters(snps)

      // unallocate sss and snps, no longer needed.  Place output in
      // tightly coupled memory instead.
      output = (double *)TC_RAM

      // process the given window
      kalman_filter()

      // send the message back to Master
      altera_avalon_mailbox_post(r)         // frame number
      altera_avalon_mailbox_post(i)         // slave ID
end
```

4. Performance

On start up all the processors initialize. The slaves go into a wait state, waiting on work from the master. The master starts by reading in the audio file stored in flash. After Initialization the master distributes work to each of the slaves, and then waits. The master waits for a response from any of the slaves announcing completion of the assigned work. The master proceeds to process the received data, which is located in the tightly coupled memory associated with the slave. After which, the slave is given the next

available work. Thus work is allocated on a first come first served basis. Table 1, shows a qualitative analysis of the different states of the processes.

Master	Slave 0	Slave 1	Slave 2	Slave 3	Slave 4
Init	Init	Init	Init	Init	Init
g:1	w: 1				
g:2		w: 2			
g:3			w: 3		
g:4				w: 4	
g:5					w: 5
p:1					
g:6	w: 6				
p:2					
g:7		w: 7			
p:3					
g:8			w: 8		
p:4					
g:9				w: 9	
p:5					
g:10					w: 10
p:6					
g:11	w: 6				
p:7					
g:12					
p:8		w: 7			
g:13					
p:9					
g:14			w: 8		
p:10					
g:15					

Table 1. Execution of the processors

6. Discussion

Profiling of the system still needs to be done in order to find bottlenecks and to assess tradeoffs. There does appear to be problems with the Altera's SOPC Builder when working with large systems with many components. After initial generation of the system it is not possible to reopen SOPC builder again. This has been tested on different PCs running different versions of Quartus II.