Functional Units with Conditional Input/Output Behavior in VLIW Processors

Marco J.G. Bekooij, Loek J.M. Engels, Albert van der Werf, Natalino G. Busá
Philips Research Laboratory, Prof. Holstlaan, Eindhoven, the Netherlands

In this paper we extend the method to deal with coarse-grain operations in static scheduled VLIW Processors as is introduced by Busá [1]. We allow functional units with a controller that does not traverse its states in a predefined way. This makes it possible to execute a function that contains a conditional construct like an if-statement as a single operation on a functional unit. This way the performance penalty otherwise caused by branch instructions is reduced. By adding a valid input and output signals the problem is circumvented that during compilation it is for this type of functional units not known when and how many samples will be consumed or produced. We will refer to these units as Conditional Input/output Units (CIUs). The operations that are executed on CIUs are called Conditional Input/output Operations (CIOs). The difference with guarded operations is that the production of a result of a CIO depends on the state of the CIU.

An example of how CIOs can be applied in a C program is shown in Figure 1. Inside the for-loop the array of input samples is read and supplied as a stream of samples to a function with the name "CIO". This function represents a CIO at the C program level. It produces its result in the variable *dout* and the valid signal in the variable *vout*. The result $dout$ is stored in the output array. The address of the location where this result is stored is only incremented when the valid output signal is true. Consequently, invalid results are overwritten in the output array. An additional input signal *vin* is supplied to the CIO in order to indicate if the current *din* sample must be processed.

```
void CIO_loop(int *Ain, int size, int *Aout){
  int idx1=0, idx2=0, din, dout, vin, vout;
  for(int i=0; i<size; i++){
    vin=(idx > size);
    CIO(Ain[idx1++], vin, &dout, &vout);
    Aout[idx2]=dout;
    idx2=idx2+vout;} // vout 1=true, 0=false
}
```

Figure 1. For-loop in C with a CIO.

In order to demonstrate the potential benefits of CIOs a bit-packing algorithm has been implemented on an application specific VLIW processor with and without CIOs. These processors were generated from a C-program with the A|RT Designer [2] tools. The implemented bit-packing algorithm is part of an MPEG2 encoder and it concatenates code-words in order to obtain a stream containing multiples of 8 bit words. We assume that the code-words and their length are stored in an array in memory. The bit-packing algorithm reads the code-words and their lengths one by one from this memory and concatenates the code words until the required multiple of 8 bits is collected and writes the result in an output array in memory.

The bit-packing algorithm is implemented using a CIO with algorithm that is similar to the one shown in Figure 1. The CIO performs the concatenation of the code words and produces a result only after the number of bits collected in an internal register of the functional unit has exceeded the bit-width of an output word.

Table 1. Bit-packer implementation characteristics. (fclk=100 MHz, 0.18 μm)

Example	II (cc)	functional units	code size (w. × l.)	cell area (μm^2)
No CIU	22	7	63 × 123	130000
With CIU	1	9	61 × 54	122646

The results shown in Table 1 demonstrate that the VLIW processor implementation with CIU has a higher performance and a smaller code size and cell area. The main reason for a smaller code size and area of the processor with a CIU is that the functional units are less complex because they need to execute fewer different operation types. The main reason for the increase in performance is the elimination of branch instructions.

References

[1] N. Busá, A. van der Werf, and M. Bekooij. Scheduling Coarse Grain Operations for VLIW processors. Madrid, Spain, 2000. ISSS.
[2] Frontier Design. *http://www.frontierd.com*.