Integrated Device Technology, Inc.

UPGRADE STRATEGIES FOR IDT79R3051™-BASED DESIGNS

APPLICATION NOTE AN-113

By Phil Bourekas

INTRODUCTION

The IDT RISController™ family includes various highly-integrated microprocessors providing high levels of performance with low system cost. Currently, the R3051™ family includes three different devices, each providing differing levels of price performance, yet each pin-compatible with each other. This allows the system designer to implement a single base system, yet offer various end products at different capability levels. The end result to the customer is reduced time to market for a product family, and the amortization of a single development effort over a wider variety of end products. This wide range of pin-compatible performance is not currently achieved by any other RISC processor family.

This application note describes system design techniques that insure a high degree of interchangeability with no real design impact.

THE R3051 FAMILY

Common characteristics of the R3051 family include high integration at low cost. All current family members are pin-compatible. All family members include:

- Substantial amounts of separate instruction and data caches integrated on-chip. Although the amount of caches varies across different family members, all devices contain enough cache on-chip to achieve extremely high performance with low-cost memory systems. The caches on the R3052 and on the R3081™ are actually larger than the cache on the Intel 80486 high-end processor, enabling these devices to offer higher performance at lower cost.
- MIPS R3000A compatible integer CPU. The R3051 family was designed by integrating cache and a low-cost bus interface around the standard MIPS R3000A CPU. This RISC core is widely recognized as an extremely high-performance execution engine, with powerful compiler and development tools. Some of the features of the core include a large register file, single cycle ALU, rich set of branch instructions (including compare operations as part of the branch), and separate, autonomous integer multiply and divide. Since the R3051 was designed using the standard core, 100% software compatibility is guaranteed. Thus, compiler tools, real-time operating systems, and other software tools developed around the standard R3000A work without modification on the R3051 family.
- Optional Translation Look-aside Buffer (TLB). The "E" (Extended Architecture) versions of the RISController family feature a 64-entry, fully associative TLB. The TLB allows virtual addresses to be translated into physical addresses on a 4kB page basis. The TLB is useful in providing memory protection and debug utilities in any application; in other applications, such as those using a real-time operating system, or in an X-windows server, the TLB allows increased system functionality to be provided.
- Simple, low-pin count bus interface. The R3051 family uses a time-multiplexed 32-bit address and data bus to communicate with memory. Internal to the processor are 4-deep read buffer and write buffer FIFO's to decouple the speed of the internal execution core from the slower speed memory system. The multiplexed bus arrangement has many advantages, such as lower-cost interface chips and ASICs, without impacting system performance.

Currently, there are three family members. These are:

- The R3051/51E. This device features 4kB of Instruction cache and 2kB of Data Cache. There is no hardware floating-point unit available on this device.
- The R3052/52E. This device features 8kB of Instruction cache and 2kB of Data Cache. As with the R3051, there is no hardware floating-point unit available on this device.
- The R3081/81E. This device introduces a number of new features to the family. The primary features of interest are changes to the caches, and inclusion of a hardware floating-point unit; other features will be described throughout this application note. The R3081 implements 16kB of Instruction Cache and 4kB of Data Cache; kernel software can dynamically reconfigure the on-chip caches as 8kB of Instruction and 8kB of Data Cache.

POTENTIAL UPGRADE OPPORTUNITIES

A number of possible system upgrades from a single, base design are possible. Elsewhere in this application note, design considerations to assure interchangeability are described.

Possible upgrade strategies include the following techniques:

Upgrading Cache Size

As all devices are pin compatible; it is possible to increase performance of an application by upgrading the amount of cache available on-chip. Thus, holding all other components the same, an R3051 may be removed and replaced by an R3052 to double the instruction cache. An R3052 can be removed and replaced with an R3081, doubling both the instruction and data caches.

Add Hardware Floating-Point

One upgrade to higher performance involves upgrading an R3051 or R3052 to an R3081 and taking advantage of the on-chip floating-point accelerator. Later in this applications note, software considerations for such an upgrade are described.

This upgrade will obviously substantially increase the performance of software containing floating-point operations; while the IDT software floating-point environment is very efficient, the floating-point unit of the R3081 dramatically outperforms integer emulation, and may result in a significant speed-up of some applications.

Increasing Frequency

Obviously, one way to increase performance is to increase the system frequency. This may or may not be easy to do, depending on the exact system design. Obviously, such an upgrade will typically require the replacement of multiple devices on the PCB.

Note, however, that R3051 family packaging insures that the same footprint and pinout is available across the full frequency range of the family, and for all of the family members. Thus, the same 84-pin PLCC footprint used for a 20MHz R3051 accommodates the package for a 40MHz R3081, even though that device consumes more power. This obviously simplifies upgrading a design to a higher frequency processor. Design techniques for increasing frequency may include:

- Using faster memory devices to achieve the same relative access time.
- Using faster control logic, such as faster PALs or transceivers, to increase set-up time and reduce propagation delays. For example, a 15ns PAL may be replaced with a 10ns PAL, effectively allowing the clock period to be reduced 5ns.
- Re-programming PALs and control logic to increase the number of wait cycles. While this will reduce the frequency normalized performance, the absolute performance will be increased substantially, since the processor will execute (typically out of its internal cache) at a higher rate.

"Clock Doubler" Operation

The R3081 presents a particularly unique opportunity to upgrade systems using an R3051 or R3052. This is particularly due to the "half-frequency bus" mode of operation of the R3081.

A dramatic system upgrade can be achieved by:

1. Removing a 20MHz R3051 or R3052 and replacing it with a 40MHz R3081.
2. Selecting the "half-frequency bus" and "1x clock" modes via the reset vectors.

The resulting system bus will continue to operate at 20MHz, but the CPU will execute out of its internal cache at 40MHz. The resulting system will typically see its performance more than double (recall that the upgrade to the R3081 will also increase the on-chip caches and add hardware floating-point, relative to the R3051 or R3052).

It is also interesting to note that the performance impact of running a 40MHz processor with a 20MHz bus is not as severe as one would intuitively guess. This is due to the fact that memory access time is really in units of time, rather than in wait states. That is, 200ns access memory is 4 clock cycles at 20MHz and is 8 cycles at 40MHz; the absolute time is not improved by running the bus faster.

Intel has estimated that for the i486 with clock doubling, running the bus at one-half the CPU execution rate is approximately 11% less efficient than running the bus at the full CPU rate on benchmarks such as the SPEC benchmark suite. The R3081 contains more than twice the amount of on-chip cache as does the i486, and thus will be even less dependent on bus performance; thus, the performance degradation should be even less.

DESIGN CONSIDERATIONS FOR UPGRADING

The remainder of this applications note details specific techniques which facilitates the interchange of various members of the R3051 family. In general, all devices are pin and footprint compatible, so there are no PCB issues to be concerned about. In general, the only things needed to upgrade a design are:

- Design it around an R3051. The R3081 does include some superset features relative to the R3051 which simplifies high-speed systems; however, if a system works for the R3051, it will work for an R3081.
- Make the software independent of cache size. The various devices include varying amounts of cache on-chip. An algorithm to determine the amount of cache available is presented in this applications note.
- Have a strategy for software floating-point versus hardware floating-point. The R3081 adds a high-performance hardware floating-point accelerator, as well as increasing the cache size. This applications note describes various software techniques for dealing with software emulation versus hardware acceleration of floating-point.

Thus, this application note details specific hardware choices and software choices which facilitate interchanging CPUs. In addition, the application note illustrates techniques for determining the presence or absence of the R3081 config register, the R3081 FPA, and the amount of cache on-chip.

SOFTWARE CONSIDERATIONS FOR UPGRADING SYSTEMS

Some of the system upgrade considerations should be accommodated in the application software (especially the kernel). It is possible to develop a single binary set of code which performs across all of the family members.

Sensitivity to Cache Size

Obviously, one characteristic difference among the various family members is the amount of Instruction and Data cache available. Thus, to insure interchangeability among these devices, the software should be written to be insensitive to the cache sizes.

Typically, very little of the actual application will be functionally sensitive to the amount of on-chip cache; the primary difference will be in the performance achieved. This is the primary advantage of caches with respect to memory mapped zero-wait state RAM; caches are transparent to the software, and do not affect the memory map.

Typically, the only part of the software that may be sensitive to the cache size will be the boot/initialization software, which may perform certain memory (including on-chip cache) diagnostics, and which must initialize the on-chip cache by performing a cache flush.

Figure 1 shows a listing of a routine to perform cache sizing. This routine uses bits of the on-chip status register to isolate the cache (to prevent writes or cache misses from propagating to memory), and to swap the cache (to perform the algorithm on the Instruction cache). In order to determine hit or miss, the algorithm places a marker in the first word of the cache, and then looking for the cache size such that a read of the cache forces a wrap-around to reading location zero. Once this occurs, the maximum cache size has been exceeded, and thus the cache size is known. Other algorithms could use the cache miss bit of the status register, rather than a marker value. This capability is provided in the IDT/kit™ and IDT/sim™ software packages from IDT.

Once the cache size has been determined, it is used in the cache flush routines (for example) to completely flush the caches. Note that if the only time the cache is flushed is at system start-up, it is acceptable to assume a worst case (large) cache size and flush that amount of cache; caches smaller than the size assumed will merely be flushed multiple times, resulting in wasted execution time but correct functionality. On the other hand, applications which perform cache flushing as part of ongoing operation (e.g. to assure cache coherency when DMA operations are used) would be sensitive to performance, and thus would desire to flush only the proper amount of cache.

Floating-Point Presence

Another difference between various family members has to do with the presence or absence of the floating-point. This distinction may have two impacts on the software environment:

- The initial setting of the coprocessor 1 usable bit should reflect whether or not a hardware floating-point is available. It is possible to create a software environment which can dynamically determine the presence or absence of the FPA.
- The actual binary executable of the application may be best optimized according to the presence or absence of a hardware floating-point. This is discussed below.

How to Determine Floating-Point Presence

There are at least two different methods for determining whether a floating-point is present. One way is to perform floating-point operations and determine whether the results are reasonable; these operations could be as simple as moving data into and out of the FPA registers to see if they are present, through performing floating-point calculations and examining the results (or even possibly seeing if an exception is reported). If the floating-point is detected as present, coprocessor 1 should be marked as usable by the kernel.

Another method would be to use the CpCond(1) (coprocessor 1 condition) flag. The hardware could tie the CpCond(1) to a known state (e.g. HIGH); software could then perform a compare operation (or move to the fp cscr register) to cause CpCond(1) to report the opposite polarity. A simple branch on coprocessor (1) condition will then determine whether the CpCond(1) signal is driven by an on-chip FPA, or by the off-chip pull-up resistor.

FPA Impact on the Binary Code

There are two methods for dealing with the software which may or may not have a hardware floating-point unit. The optimal method depends on trade-offs between a single binary set operating either with or without a hardware FPA, versus a single source set compiled twice resulting in two binaries (one targeted to a hardware FPA and one targeted to an integer only environment).

Using a Single Binary with and Without an FPA

If the system designer chooses to implement a single binary capable of taking advantage of a hardware FPA when one is available, all that needs to be done is to tap into the inherent capabilities of the MIPS coprocessor architecture. Specifically, if the kernel marks the coprocessor 1 FPA as unavailable, FPA instructions will cause a trap to occur. The kernel can then perform an integer interpretation of the FPA instruction. The application software is then compiled to assume the availability of a hardware FPA: if one is available in the system fine; if not, traps will occur when FPA operations are encountered, and the kernel can perform an emulation of the function.

Using this technique requires two things in the software:

- Boot software must perform the diagnostics described above to determine the appropriate setting for the coprocessor 1 usable bit.
- The kernel must include the capability to emulate the entire FPA unit, including the FPA operations, the register file, and the FPA exception mechanisms used by the application.

While this technique has the advantage of resulting in a single binary which works in either environment, the result is added complexity and a loss of performance in the environment in which no FPA is available. Specifically, the kernel must provide an emulation library of the entire FPA; and, software FPA operations will include additional overhead from the CPU exception model and from emulating all aspects of the FPA, even though a given operation only requires a subset of the FPA functionality.

Developing Two Binaries from a Single Source

Another technique exists whereby two distinct binaries are developed from a single source tree. Each of the resulting binaries is fully optimized for either an integer only environment, or for an environment in which a hardware floating-point is available.

This is accomplished by taking advantage of the software floating-point library capabilities of the IDT/c™ environment. IDT/c includes a compile time flag which can be used to control whether hardware FPA instructions (coprocessor 1 instructions) are generated, or whether direct calls to a software floating-point library are generated. Thus, software floating-point is not forced to emulate the register set and data type conversions of the hardware FPA, and execution is not forced to go through the CPU exception model. The resulting binary operates much more efficiently than one which goes through the trap and emulation model described above.

A separate applications note describes how to determine the optimal compilation environment for a given application.

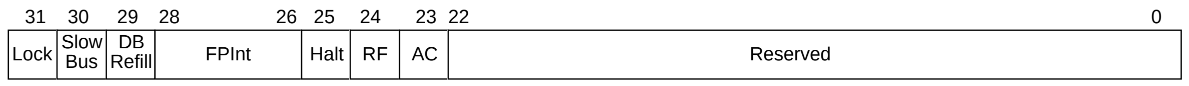
```
/*************************************************************************
**
** _size_cache()
** returns cache size in v0
**
*************************************************************************/

FRAME(_size_cache,sp,0,ra)
      .set        noreorder
      mfc0        t0,C0_SR                  /* save current sr */
      and         t0,~SR_PE                 /* do not inadvertently clear PE */
      or          v0,t0,SR_ISC              /* isolate cache */
      mtc0        v0,C0_SR
      /*
       * First check if there is a cache there at all
       */
      move        v0,zero
      li          v1,0xa5a5a5a5             /* distinctive pattern */
      sw          v1,K0BASE                 /* try to write into cache */
      lw          t1,K0BASE                 /* try to read from cache */
      nop
      mfc0        t2,C0_SR
      nop
      .set        reorder
      and         t2,SR_CM
      bne         t2,zero,3f                /* cache miss, must be no cache */
      bne         v1,t1,3f                  /* data not equal -> no cache */
      /*
       * Clear cache size boundries to known state.
       */
      li          v0,MINCACHE
1:
      sw          zero,K0BASE(v0)
      sll         v0,1
      ble         v0,MAXCACHE,1b

      li          v0,-1
      sw          v0,K0BASE(zero)           /* store marker in cache */
      li          v0,MINCACHE               /* MIN cache size */

2:    lw          v1,K0BASE(v0)             /* Look for marker */
      bne         v1,zero,3f                /* found marker */
      sll         v0,1                      /* cache size * 2 */
      ble         v0,MAXCACHE,2b            /* keep looking */
      move        v0,zero                   /* must be no cache */
      .set        noreorder
3:    mtc0        t0,C0_SR                  /* restore sr */
      j           ra
      nop
ENDFRAME(_size_cache)
      .set        reorder
```

Figure 1. Cache Sizing Software



Lock:	1 -> Ignore subsequent writes to this register
Slow Bus:	1 -> Extra time for bus turnaround
DB Refill:	1-> 4 word refill
FPInt:	Power of two encoding of FPInt <-> CPU Interrupt
Halt:	1 -> Stall CPU until reset or interrupt
RF:	1 -> Divide frequency by 16
AC:	1 -> 8kB per cache configuration
Reserved:	Must be written as 0; returns 0 when read

Figure 2. R3081 Config Register

The method of dealing with floating-point operations in an integer CPU only environment is particularly important in the evaluation of a compiler platform; techniques such as the "mix and match" approach supported by IDT/c allows the best capabilities of the MIPS compiler toolchain to be integrated with efficient software floating-point emulation.

The obvious advantage of this approach is the optimum performance achieved for both the integer only system and the R3081-based (hardware FPA) system. Using distinct EPROM sets at manufacturing time, or upgrading both the EPROMs and processor as a field upgrade, are obvious consequences, but in general are not particularly onerous (EPROM upgrade can be a replacement of EPROMs, or, for FLASH EPROM, a re-programming of the EPROMs resident on the board).

The R3081 Config Register

The R3081 includes, as part of coprocessor 0, an additional control register called "Config". The R3081 Config Register is shown in Figure 2.

The Config register controls various aspects of system functionality. If these features are used in an R3081 system, software must first determine whether they are available.

To determine whether the current device is an R3081 (and thus whether the config register is available), software can use various techniques. One straightforward technique is to determine whether or not there is an FPA; if so, the device is an R3081. Similarly, software could determine the cache sizes available, and see if these correspond to the organization the R3081.

Other techniques are also possible; for example, size the cache, then reconfigure the cache by writing to the config register; re-size the cache to determine that the change occurred. Obviously, if the change occurs, the config register is available.

Note that writes to this register location in the R3051 or R3052 will have no effect; no side effects occur, and no traps are signalled. Reads of the config register produce an undefined data result for the R3051 and R3052.

If the config register is used when an R3051 is in place, various other considerations exist. These are:

- *Floating Point Interrupt.* In general, if an R3051 application intends to also work with an R3081, one of the CPU interrupt inputs needs to be reserved for the hardware FPA of the R3081. The default interrupt is Int(3), but the config register allows a different interrupt assignment to be used. The corresponding interrupt input pin of the R3081 is then ignored. Thus, the PCB should contain a pull-up resistor at the interrupt pin; when an R3051 is used in the application, no interrupt will be signalled.
- *Reduced Frequency.* This mode dramatically reduces the power consumption of the R3081, by reducing its operation frequency. This mode is unavailable in the R3051. In general, the only real functional system change that occurs is that the SysClk output clock frequency is also reduced; thus, if DRAM refresh, for example, was derived from this clock, the counter value should be reprogrammed. If an R3051 is told to "reduce frequency", nothing will happen.
- *Halt.* This control bit forces the R3081 to stall until an interrupt input is asserted, or a reset is encountered. This mode is unavailable in the R3051, and no simple software equivalent exists.
- *Data Block Refill.* The R3081 allows the block size read on a data cache miss to be dynamically reconfigured by software. The initial value is set by the reset value. In general, this bit may affect the performance of software, but is unlikely to impact its functionality.
- *Alternate cache.* This bit allows the caches to be dynamically reconfigured for the R3081. A cache flush should be performed after the cache is reconfigured. An earlier section of this applications note discussed how to make software independent of the cache organization.
- *Lock.* This bit allows software to inhibit subsequent writes to the Config register. Thus, boot software can set up the operation mode, and then protect it from other software.
- *Slow Bus Turnaround.* This bit allows systems to enjoy longer time between A/D bus mastership transitions. However, this software control is not available on the R3051. If the system designer desires extra time, and also desires to be able to interchange R3051s and R3081s, the hardware technique described in applications note AN-97 is appropriate. This technique uses the DMA arbiter interface of the CPU to insure that new transactions are not begun until ample time for bus turn-off has passed. This hardware technique works equally well with both the R3051 and R3081.

HARDWARE DESIGN ISSUES

There are various hardware design considerations that may impact the ability to interchange various members of the CPU family. With proper design, these considerations can be dealt with no real system impact.

Slow Bus Turn

Bus turn is the amount of time allowed to change mastership on the A/D bus of the processor. In general, a read followed by a write can cause a change in bus direction in one-half bus cycle. At 33MHz, this is 15ns.

The system designer may implement an architecture which, by using appropriate transceivers and control signals, can tolerate a rapid bus turn. Alternatively, the designer may desire to increase the minimum amount of time.

Although the R3081 includes a bit in the Config register to slow the bus, this technique does not work with the R3051. Instead, the hardware technique of using BusReq to insure a longer tri-state time is recommended. This technique is described in applications note AN-97.

Coherent DMA

The R3081 includes a hardware interface to insure cache-coherency in systems using DMA. This interface is unavailable in the R3051.

Many MIPS applications perform multi-master cache coherency via software techniques, and thus do not require hardware-based coherency. While hardware-coherency will improve the performance of some applications, relying on software (which may, for example, flush the entire data cache once a DMA operation is completed to insure coherency. This technique will function equally well with either the R3051 or R3081.

Floating-Point Interrupt

The R3081 uses one of the interrupt input pins to report exceptions to the CPU. The hardware should reserve one of the input pins for this function, and provide logic or pull-up resistors to insure that this input is held HIGH for an R3051 or R3052.

CpCond(1)

The R3081 uses this input to report the results of comparisons back to the CPU; thus, the external input pin is ignored. R3051 systems should provide a pull-up resistor for this pin. Earlier in this applications note, a method to use this pin to determine the presence or absence of an FPA was described.

Reset Mode Vectors

Both the R3051 and R3081 use the same basic technique to perform reset mode selection of various options. Figure 3 illustrates the mode vector logic for the R3081. Note that for the R3051, Int(5:3) mode vectors are reserved, and must be held HIGH during reset.

Options include:

- *Tri-state.* This option is used to perform board testing, and is available in all devices.
- *BigEndian.* This option selects the data byte ordering convention, and is available in all devices.
- *Data Block Refill.* This option selects single versus four-word refill on data cache misses. Although this option is available in all devices, software (via the config register) can dynamically change the value for the R3081.
- *Coherent DMA Enable.* This option enables the coherent DMA interface of the R3081. For the R3051, this input must be HIGH at reset.

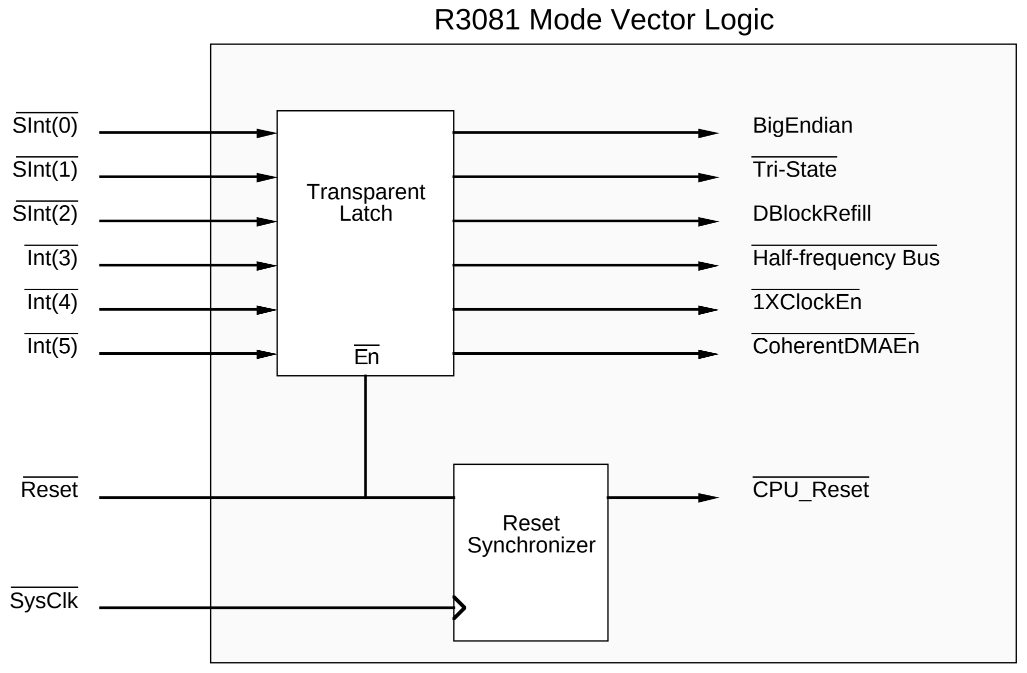

R3081 Mode Vector Logic
$\overline{\text{SInt(0)}}$
$\overline{\text{SInt(1)}}$
$\overline{\text{SInt(2)}}$
$\overline{\text{Int(3)}}$
$\overline{\text{Int(4)}}$
$\overline{\text{Int(5)}}$
Transparent Latch
$\overline{\text{En}}$
BigEndian
$\overline{\text{Tri-State}}$
DBlockRefill
$\overline{\text{Half-frequency Bus}}$
$\overline{\text{1XClockEn}}$
$\overline{\text{CoherentDMAEn}}$
$\overline{\text{Reset}}$
$\overline{\text{SysClk}}$
Reset Synchronizer
$\overline{\text{CPU\_Reset}}$

Figure 3. R3081 Mode Vector Assignment

- *1x Clock Mode.* This option instructs the R3081 that the input clock provided is at the CPU operation frequency, rather than at twice the frequency. In the R3051, only the "2x" clock is available, and this vector must be held HIGH.
- *Half-frequency Bus.* This option instructs the R3081 to operate its bus interface at one-half the execution rate. This option is unavailable in the R3051, and must be held HIGH at reset.

In order to design a system to accommodate either an R3051 or R3081, it may be desirable to include jumpers for the R3081-only options. Thus, when an R3081 is included in the design, various of the hardware options may be changed. This may open up other upgrade strategies, such as the clock doubling capability described earlier.

SUMMARY

By following a few simple rules, the system designer can implement a base R3051 system which can easily upgraded to higher performance. Upgrade options include more amounts of cache on-chip, the addition of hardware floating-point, and increases of frequency. With the R3081 half-frequency bus mode, the operation frequency of the execution engine can be substantially increased while maintaining the same (or even slower) bus interface frequency.

Thus, the IDT RISController family effectively reduces the time to market of new product families, and maximizes engineering return on investment by enabling one design effort to result in multiple end products.