

Compilation Strategies: Alternate Approaches to Achieving Low Power Consumption



by Clyde Stubbs, HI-TECH Software

The increasing emphasis on green technologies has focused attention on power consumption in all devices, even those that are connected to a main power supply. The main source of power consumption is switching transients which occur every time a digital output changes state. Thus, reducing the number of transitions, by reducing the total number of instructions, results in an immediate reduction in the amount of power consumed. Most microcontrollers have sleep modes that reduce power consumption to a level that approaches the leakage current of the device.

The objective in designing software for low power is to make use of the deepest sleep state the application allows, and to spend as much time as possible in that state. Compilers can have a subtle but significant effect on how much time the microcontroller spends in sleep mode. All too often, the method of compilation wastes CPU cycles on interrupt routines and, in devices with banked memory, on locating addresses in memory.

Interrupt Routines

The compiler's contribution to interrupt latency is in the number of registers it saves in response to an interrupt, or the context. In order to prevent memory overwrites, conventional compilers save every register that might be used by an interrupt. Most compilers have no way of knowing which registers will or will not be used by a given interrupt, so they save them all. However, saving more registers than necessary also consumes more power than necessary, while also putting a drag on performance by increasing interrupt latency.

For example, a conventional compiler for Microchip's PIC16 always saves 8 bytes of data for every interrupt, regardless of what data will be required. Saving 8 bytes of data requires a total of 42 instruction cycles (23 for the

context save and 19 for the restore). This may not seem like much, but in an interrupt intensive application, the CPU could spend thousands of extra cycles "awake," consuming power unnecessarily.

Newer compilers are now available with omniscient code generation (OCG) technology that has the intelligence to save only those registers that are required for each particular interrupt. Omniscient code generation works by collecting comprehensive data on register, stack,

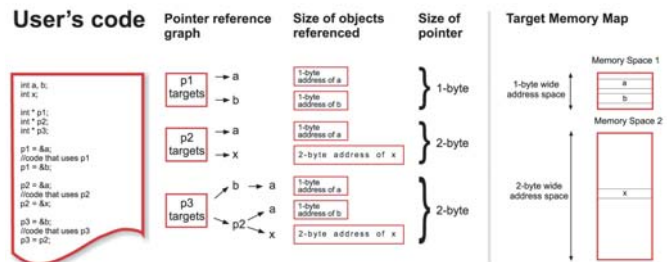


Figure 1. Pointer reference graph.

pointer, object and variable declarations from all program modules before compiling the code.

An OCG compiler combines all the program modules into one large program that it loads into a call graph structure. Based on the call graph, the OCG code generator creates a pointer reference graph that shows each instance of a variable having its address taken, plus each instance of an assignment of a pointer value to a pointer (either directly, via function return, function parameter passing or indirectly via another pointer). It then identifies all objects that can possibly be referenced by each pointer. This information is used to determine exactly how much memory space each pointer will be required to access.

The OCG compiler knows exactly which functions call and are called by other functions, which variables and registers are required, and which pointers are pointing to which memory banks. This gives the compiler the necessary intelligence to know exactly which registers will be

Power Sources & Circuit Protection

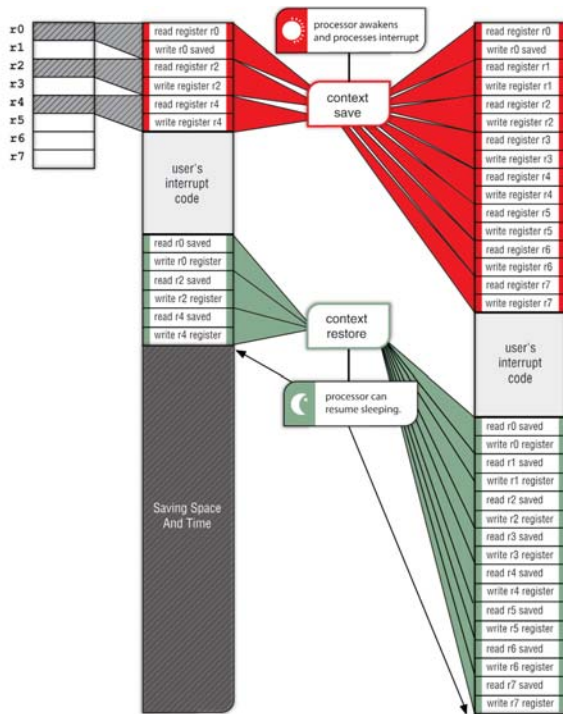


Figure 2. Relative context sizes for OCG and non-OCG compilers.

used for every interrupt in the program so it can generate code accordingly, minimizing both the code size and the cycles required to save and restore the context.

The compiler determines the size of the context for each interrupt dynamically, depending on the state of the code at the time of compilation. An interrupt may require as few as 17 cycles: 10 to save the context, and 7 to restore it. In the worst case, the OCG compiler uses only 25 cycles for the context save and restore. Compared to a conventional compiler, an OCG compiler can reduce the number of interrupt-related instruction cycles by 40 percent to 60 percent.

Depending on the application, the cycle savings can be substantial. An interrupt-driven serial communication port with a baud rate of 480,600 bps generates 24,000 interrupts per second. Using a conventional compiler with 42 instruction cycles per interrupt (168 clock cycles per interrupt), saving and restoring the context will use over 4,032,000 CPU cycles per second or 20 percent of the available cycles on a 20 MHz PIC16. An OCG compiler, averaging 21 instruction cycles per interrupt (84 clock cycles per interrupt), can reduce that number to only 2,016,000

Power Sources & Circuit Protection

cycles, saving the clock cycles otherwise spent on saving and restoring contexts, and allowing the CPU to be put into sleep mode for 10 percent of its cycles. Assuming 10 mA active and about 1 μ A sleep mode power consumption, an OCG compiler could reduce total MCU power consumption by nearly 1 mA, or about 10 percent.

Banked memory architectures can waste cycles.

Many 8- and 16-bit microcontrollers have banked memories that cannot be addressed simultaneously. Switching between the memory banks requires at least two bank selection instructions. If data in one bank must be written to another bank, bank selection instructions are always necessary. Obviously, placing all the variables accessed by a function in the same memory bank will reduce the number of bank selection instructions and the total required cycles for the application.

Since an OCG compiler knows every register, stack, pointer, object and variable declaration from all program modules, it can optimize every variable and register allocation, as well as the size and scope of every pointer and every object stored on the compiled stack. It can allocate

memory optimally to minimize or eliminate power-hungry bank selection instructions.

The total number of cycles that can be saved is extremely application-dependent, and therefore, difficult to quantify. It is possible that omniscient code generation could reduce the total cycles required by 30 percent to 50 percent, with a linear effect on MCU power consumption.

Conclusion

The way in which the compiler manages interrupts and memory usage can have a significant impact on power consumption. Newer compilers with omniscient code generation technology can make a substantial contribution to saving cycles and power.

Clyde Stubbs is founder and CEO of HI-TECH Software, Queensland, Australia. His university research in compiler technology led to the founding of the company in 1984. In 2006, he developed omniscient code generation (OCG).
