Instrumentation Control Using the Rabbit 2000 Embedded Microcontroller

Ian S. Schofield*, David A. Naylor
Astronomical Instrumentation Group, Department of Physics, University of Lethbridge,
4401 University Drive West, Lethbridge, Alberta, T1K 3M4, Canada

ABSTRACT

Embedded microcontroller modules offer many advantages over the standard PC such as low cost, small size, low power consumption, direct access to hardware, and if available, access to an efficient preemptive real-time multitasking kernel. Typical difficulties associated with an embedded solution include long development times, limited memory resources, and restricted memory management capabilities. This paper presents a case study on the successes and challenges in developing a control system for a remotely controlled, Alt-Az steerable, water vapour detector using the Rabbit 2000 family of 8-bit microcontroller modules in conjunction with the MicroC/OS-II multitasking real-time kernel.

Keywords: Embedded processor, Rabbit, instrument control, MicroC/OS-II

1. INTRODUCTION

The Astronomical Instrumentation Group (AIG) of the University of Lethbridge's Department of Physics has been designing instruments for use in infrared and (sub)millimetre astronomy for over twenty years, with an emphasis on Fourier transform spectroscopy. Historically, these instruments have been driven by control software hosted on standard desktop personal computers (PCs). This approach has been highly successful, allowing for rapid and inexpensive system development using widely available software development tools and low cost, commercial off-the-shelf hardware.

In the fall of 2001, the AIG began work on a remotely controlled atmospheric water vapour detector called IRMA (Infrared Radiometer for Millimetre Astronomy). IRMA mechanically consists of a shoebox-size detector system attached to an Alt-Az motorized fork mount, which allows it to point to any position in the sky, and is attached to the end of an umbilical cable, through which it receives its power and network connection. When deployed, an IRMA unit will be mounted on each (sub)millimetre telescope antenna making up an antenna array, allowing it to measure the amount of precipitable water vapour in each antenna's line of sight. The resulting data will then be used to correct phase distortion contained in the signal data and thereby increase the angular resolution of the telescope.[1] IRMA is being developed with the goal of performing phase correction for the Atacama Large Millimeter Array (ALMA), which will be the world's most powerful ground-based telescope, when completed early in the next decade.

In order to make IRMA a feasible instrument for the ALMA project, it's hardware and software architecture has to meet the following requirements:

- Stand-alone, outdoor operation under highly variable, high-altitude (5000 m) weather conditions: average temperatures at Chajnantor in 2003 ranged from –10°C to 15°C .[2]
- Highly stable, real-time multitasking operating system with plentiful digital input/output (DIO) and serial I/O lines for instrument control and data acquisition.
- Ethernet connection to transfer collected data.
- Shoebox-sized detector compartment (38 cm x 22 cm x 18.5 cm), of which a small portion is available to install a computer board.
- Power consumption of the entire IRMA system must not exceed amount of power produced by a modest-sized solar voltaic power generation system.
- Low price for production of ~60 or more units.

* ian.schofield@uleth.ca; phone 1-403-329-2426; fax 1-403-329-2057; www.uleth.ca/phy/naylor/

An embedded processor was an obvious choice over a PC-based control and data-acquisition solution to meet these requirements. After careful review of embedded control products, the Rabbit 2000 microcontroller was selected, having met these requirements, as well as offering an affordable, feature-rich software development package with customer support.

This paper discusses our experience using the Rabbit Semiconductor's Rabbit 2000 based microcontroller modules to perform IRMA's control and data-acquisition, considering their strengths and weaknesses in relation to PC-based systems. The paper is structured as follows: The technical specifications of the Rabbit 2000 microcontroller modules are introduced in Sec. 2. The architecture and development of the hardware and software for IRMA is described in Sec. 3. General conclusions about the design of hardware and software with the Rabbit 2000 microcontroller modules are drawn in Sec. 4.

2. RABBIT 2000 MICROCONTROLLER MODULES

The Rabbit 2000 family of microcontroller modules come in a variety of configurations. In essence, they all consist of a credit card-sized multi-layer printed circuit board containing a surface mounted Rabbit 2000 microprocessor, static random access memory (SRAM), flash memory, a 10 megabit/second Ethernet controller (in the network-equipped models) and two rows of DIO pins on the reverse side of the board. IRMA uses two different models of Rabbit microcontroller modules: the network-equipped RCM2100 (in Fig. 1), used for handling communication and instrument control, and the RCM2010, used for controlling the Alt-Az mount. Rabbit 2000 microcontroller modules can support as much as 1 megabyte of memory (equally divided between SRAM and flash memory), 40 DIO lines, and microprocessor clock speeds as high as 25 MHz. Specifications for the RCM2100 and RCM2010 are given in Table 1.

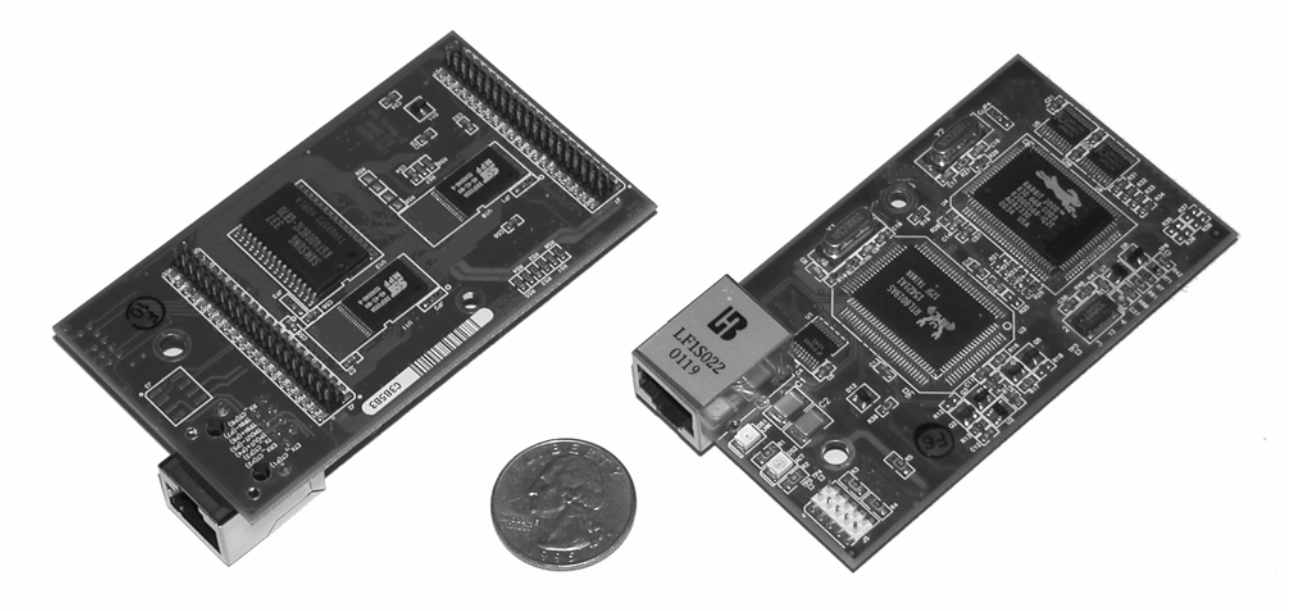

Fig. 1. Rabbit 2000-based RCM2100 Microcontroller Module (reverse and top sides).

Rabbit 2000 microprocessors are descendents of the venerable Zilog Z-80/Z-180 architecture, from which they inherit a great degree of machine-level instruction compatibility with the Z80 and related processors, sharing a similar register layout, memory addressing modes and machine instructions. The primary difference between the two processors is that the Rabbit's instruction set is optimized for handling 16-bit operations, such as 16-bit arithmetic and memory manipulation, in order to make the processor more compatible with C language compilers. The Z80/Z180 instruction set, which does not have efficient 16-bit instructions, requires larger, more inefficient routines to perform similar tasks.

Being an 8-bit microprocessor, the Rabbit 2000 can only directly access a 16-bit address space (64K), but through the use of extended memory, can access one megabyte of memory.

FEATURE	RCM2100	RCM2010
Microprocessor	22 MHz Rabbit 2000	25.8 MHz Rabbit 2000
Ethernet	10Base-T, RJ-45	None
Memory: Flash	512K	256K
Memory: SRAM	512K	128K
DIO	40 Lines (grouped into five 8-bit ports, shared with 4 serial ports). Less lines available if networking used.	40 Lines (grouped into five 8-bit ports, shared with 4 serial ports).
Serial IO	Four 5V CMOS-compatible	
Real Time Clock	Yes	
Timers	Five 8-bit timers and one 10-bit timer	
Connectors	Two 2 x 20, 2 mm IDC headers	
Power	5 V, 140 mA	5 V, 120 mA
Board Size	89 x 51 x 22 mm	58 x 48 x 13 mm

Table 1. RCM2100 and RCM2010 specifications.[3,4]

Flash memory is used to permanently store the executable code and any related static data, such as constants, tables or files, while the static RAM holds volatile data created during runtime. Thus, rabbit programs cannot be larger than the available flash memory. Although 512K (roughly equivalent to 50,000 C statements) may not sound like a great deal of memory, it is more than adequate for running serious control and data acquisition programs, because the Rabbit's C language compiler, Dynamic C, produces lean and efficient executable code. Nonetheless, limited memory can be an issue when developing software for any embedded system, especially if the desired application needs to log data. A major difference between developing software for the Rabbit 2000 and desktop computers is the fact that the programmer does not have available limitless memory or hard disk space.

3. CASE STUDY: IRMA

IRMA is a compact, remote controlled infrared radiometer, targeted for use in (sub)millimetre astronomy to correct phase distortion of incoming celestial signals caused by rapid variations of water vapour in the Earth's atmosphere. To be used in conjunction with a (sub)millimetre telescope antenna array, each IRMA unit will measure the amount of precipitable water vapour (pwv) in the column of atmosphere along each antenna's line of sight. These data allow to correct for phase errors contained in each antenna's signal data, thus increasing the resolving power of the array.[1]

3.1. IRMA Hardware

Operationally, IRMA must perform three fundamental tasks: process commands from the operator, control the hardware components, and store data for retrieval. Due to the amount of processing power involved in performing these tasks and the need to archive the collected data, it was necessary to divide IRMA's computing tasks among three computers: the command processor (CP), master controller (MC) and Alt-Az controller. The structure of the IRMA system is shown in Fig. 3. The IRMA CP interprets command scripts into low-level instructions, which it sends to the MC as binary command packets, receiving and storing any data returned by the MC. Since IRMA has to store data in quantities that Rabbit microcontrollers cannot handle, the CP is based on a PC running the Linux operating system. Both the MC and Alt-Az controller are based on Rabbit microcontroller modules and are dedicated to hardware control, while the CP handles communication and data archiving.

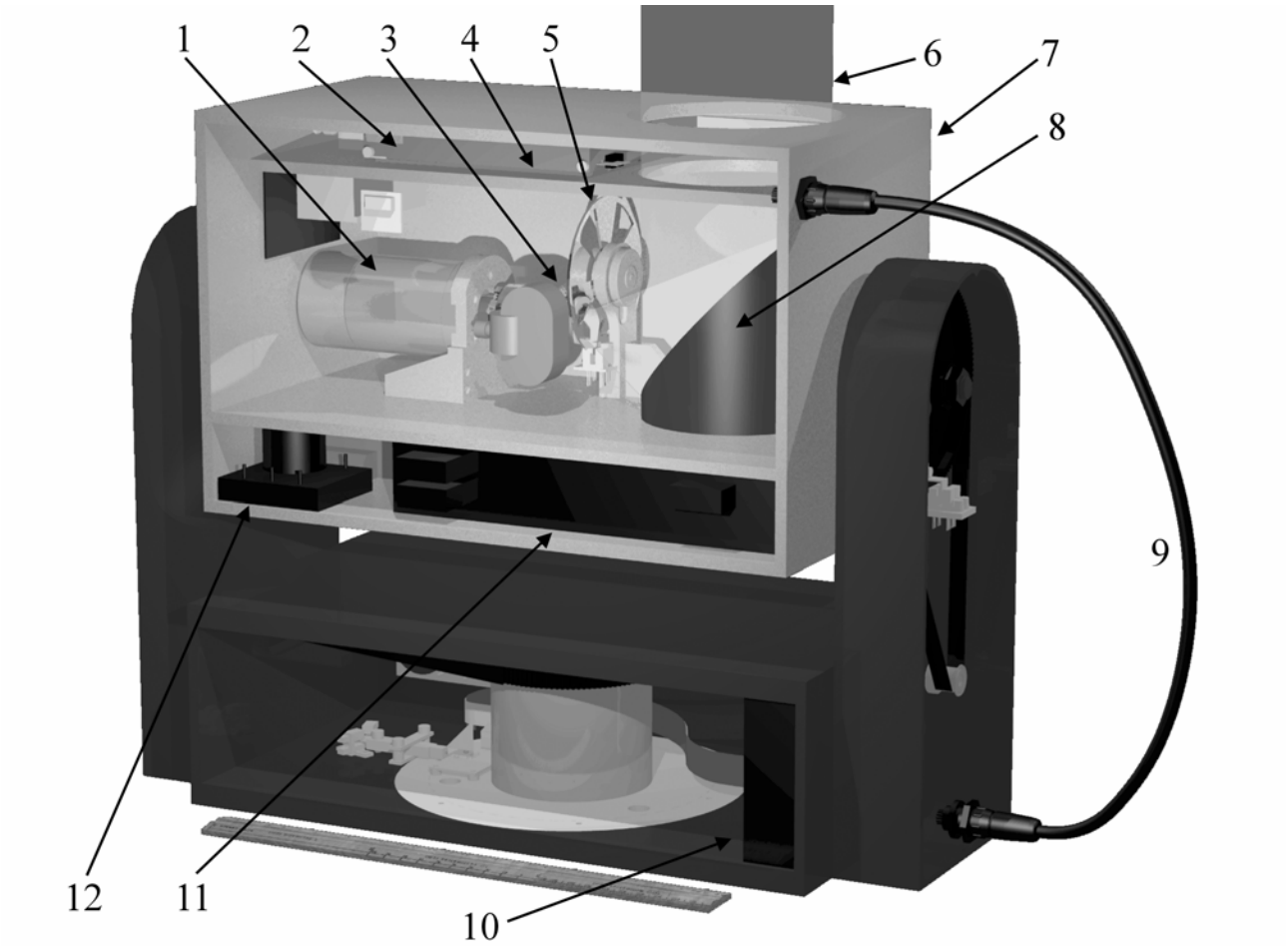

Fig. 2. Cut away view of IRMA in its Alt-Az mount.[1] 1) Cryo cooler 2) Shutter 3) MCT detector 4) Black body and heater 5) Reflective chopper 6) Input beam 7) Main board and IRMA master controller (behind detector box) 8) Parabolic mirror 9) Power/comm. umbilical 10) Alt-Az controller 11) Cryo cooler controller 12) Power supplies

The IRMA MC is based on a RCM 2100 Rabbit microcontroller unit that is linked to nine separate devices over DIO and serial channels. Each of IRMA's mechanical components are identified in Fig. 2. Infrared (IR) radiation emitted by atmospheric water vapour is chopped with a 5-blade reflective chopper wheel rotating at 455 Hz to improve its signal to noise ratio. A single notch on the wheel's perimeter triggers analog to digital (A/D) conversions on a specific blade to eliminate uncertainties associated with blade to blade differences in emission/reflectance. The notch signal is sent to the MC's external interrupt line, which triggers an interrupt service routine to perform A/D conversions. The chopper wheel, controlled by a Maxon 1-Q-EC digital motor controller, is fixed at a single rotation speed, and is enabled or disabled by the MC over a single DIO line. The IR radiation is detected by a Mercury Cadmium-Telluride (MCT) photoconductive detector cooled to below liquid nitrogen temperatures ($\leq$77 K) by a Hymatic NAX025-001 closed-cycle Stirling cycle cooler, and is controlled by the MC over a 4800 bits/second (bps) serial connection. The resulting AC signal from the MCT detector can be passed through 60 Hz or 120 Hz notch filters, which are selectable by two DIO lines to the MC.

The analog signals are digitized with a Cirrus CS5534 24-bit Delta-Sigma 4-channel analog to digital converter (ADC), which the MC controls by sending and reading serial data over the ADC's 4-wire DIO interface. The ADC samples eleven separate analog sources: eight temperature sensors (fed into the ADC by means of an 8-channel analog multiplexer), one relative humidity sensor, one atmospheric pressure sensor, and the IR detector signal.

The IRMA IR detector housing consists of a sealed aluminum box with a motorized retractable shutter. The shutter is controlled by means of a single DIO enable line and two DIO limit lines that indicate the shutter state. Opening or closing the shutter is triggered by the MC clearing or setting the shutter's enable line. Electronics on the IRMA main electronics board disables the motor once the shutter breaches either the open or closed optical limit switches. On the inside surface of the shutter (facing the inside of the box) is a black body source, used for system responsivity calibration. This black body is heated with a thin film electrical heating element. The MC enables or disables the heater via a single DIO line. Black body temperature is sensed by two of the eight temperature diodes distributed throughout the detector box.

Accurate time and position data is provided by a GlobalSat ER-101 GPS board, which feeds NMEA-formatted GPS data strings every second over a 4800 bps serial link. The MC reads the GPS in order to synchronize its own on-board real time clock (RTC) with the GPS time stamp.

Two DIO lines are reserved to monitor over current conditions on the 12 and 24 V power supplies. Although not used in the current IRMA model, it will be used in future versions.

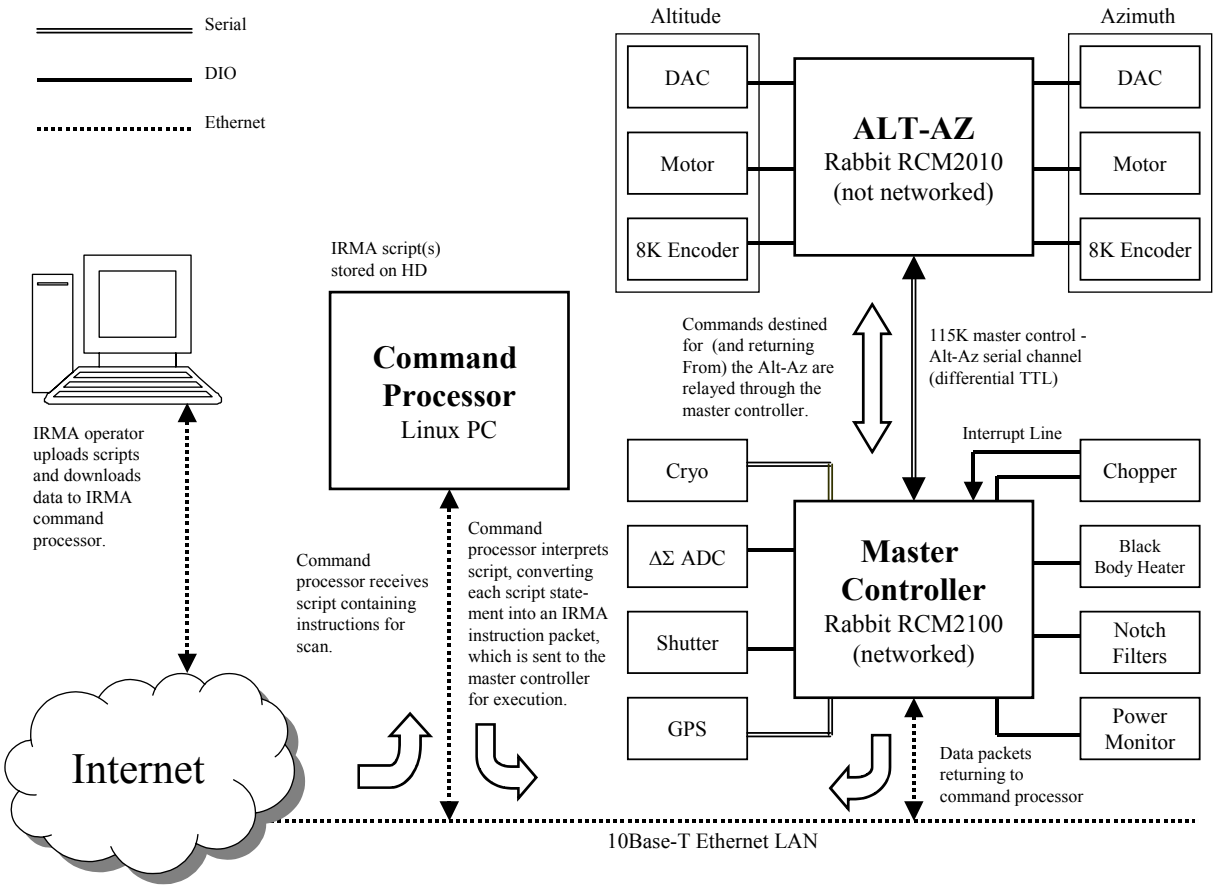

Fig. 3. IRMA control software block diagram.

The IRMA Alt-Az controller, based on a RCM2010 Rabbit microcontroller core, handles the Altitude-Azimuth mount's motor control and communications functions. Communication with the MC is carried over a 2-wire 115,200 bps serial connection using a simple ASCII string protocol with cyclic redundancy check error detection. To increase reliability, the MC - Alt-Az serial link is implemented using a RS-485 differential TTL driver-receiver pair.

The Alt-Az controller drives two Maxon EC167129 low-noise, 50W, brushless DC motors in order to articulate IRMA's two-axis Alt-Az mount. The Alt-Az processor controls each axis through an individual Maxon 1-Q-EC digital motor controller, allowing the Alt-Az to enable/disable each motor via a single DIO line. To produce motor movement, the Alt-Az controller injects a voltage level, produced by a Maxim MAX5223 8-bit 2-channel serial digital to analog converter (DAC), to the motor control units' speed input. Motor speed, which is directly proportional to input voltage going into the motor control unit, is selected by the Alt-Az controller by writing a specific 8-bit value ranging between 0 to 255 into the serial DAC.

Axis position information is read by dual US Digital E6M 2048-line optical encoders, each located on their respective axes. A dual-channel US Digital LS7266 quadrature counter chip interfaces the optical encoders to the Alt-Az controller. Controlled over 8 DIO data channels plus 4 DIO control channels, the LS7266 enhances encoder resolution by 4 times when operated in quadrature mode, allowing positions to be resolved with 8192 encoder positions per revolution (approximately 2 arc minutes per step).

3.2. IRMA Software

IRMA must be able to handle external communications in parallel with data collection and Alt-Az positioning. Additionally, IRMA must ensure certain activities (namely data sampling) are executed at precise intervals regardless of the CPU's workload, requiring that the IRMA software be multi-tasking and real-time. The Rabbit software development system, Dynamic C[5], provides a preemptive multitasking, real-time kernel, called MicroC/OS-II[6], to develop real-time multitasking programs on the Rabbit.

Preemptive multitasking involves a scheduler running in the background, invisible to the executing program, which divides processor time among a number of individual programs (or code blocks) called tasks. Each task is given a priority, which the scheduler uses to decide which task should execute at a given time out of a set of tasks waiting to be run. If a higher priority task is scheduled to run at a certain time and a lower priory task is executing, the scheduler will preempt the lower priority task (put it to sleep) and run the higher priority task instead. Once the higher priority task has completed executing, the scheduler will resume executing the lower priority process. Being real-time, MicroC/OS-II will ensure that programs running under it do not violate user-defined time constraints, regardless how busy the processor is. Hence, if a high priority task must trigger an A/D sample at some prescribed interval, the scheduler will preempt the currently running task to satisfy this requirement.

Once IRMA's functionality was identified, it was divided into three tasks, each having a particular priority level based on its ability to be preempted (see Fig. 4). The task named "metronomeTask" triggers the data collection interrupt service routine (ISR) every 200 ms by enabling the external interrupt line mapped to the chopper notch. When enough samples have been collected to form a data packet, which occurs roughly every four seconds, the metronome task signals the job task to write the latest collected data over the network to the CP. The black arrows emanating from the metronome task in Fig. 4 indicate that this task explicitly activates other tasks.

The lowest priority task, called the dispatcher task, waits for commands from the CP, and processes them depending on their expected duration. Short-duration tasks get serviced immediately while long-duration activities are passed to the job task, allowing the dispatcher to return to listening to the network for new commands. The dispatcher task is not triggered directly by the metronome task because commands do not have critical timing requirements. Rather, the task runs in any remaining time after all the higher priority tasks have executed. It is critical to put a task to sleep whenever it needs to wait for some event to occur, thereby forcing a context switch, which causes the real time kernel to execute the next task of lower priority, if one is ready. Following this sequence, eventually every task in the program will be allotted processor time to execute. Figure 5 shows a timeline of each of the parallel running tasks in the IRMA MC software, demonstrating its preemptive multitasking behaviour

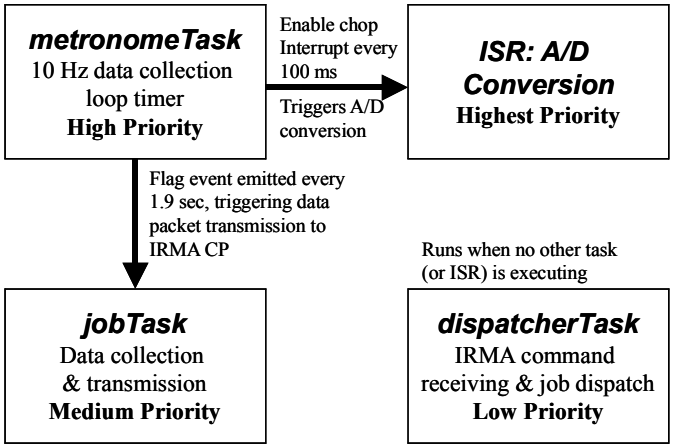

Fig. 4. IRMA master control software task structure.

Lower priority tasks whose activity duration exceeds the current amount of slack time are pre-empted by high priority processes, then resume where they left off when the processor is again available. The fact that preempted tasks can resume, remembering its state before being pre-empted, is a powerful feature of the real time kernel. Dynamic C's default method of multitasking, which involves the use of costates, forces the programmer to write code to keep track of the state of each activity in each costate (costates are similar to tasks) using state machines, which, depending on the complexity of the program, can become very elaborate.

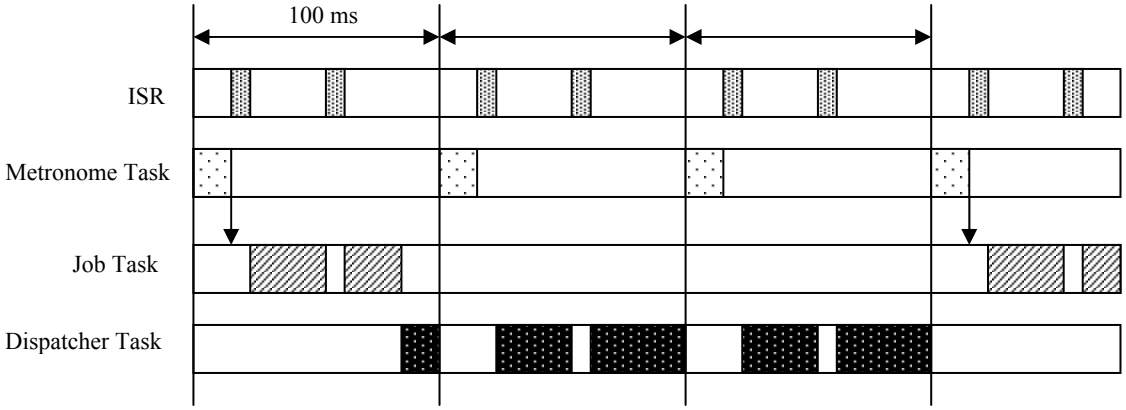

Fig. 5. Multitasking timeline of the IRMA task structure (Shaded blocks represent task activity).

Control software running on the Alt-Az controller is analogous to the MC control software: it acts as a passive server, waiting for instructions from the MC, executing short-duration commands immediately, and dispatching long-duration functions to another task in order to perform communication and motion control in parallel. Like the MC, the Alt-Az software uses the MicroC/OS-II real-time kernel to handle multitasking. The task structure consists of four tasks: a high priority task to handle communication, a lower priority task to handle long-duration motion control requests, and two low-priority motion control tasks for each axis. The motion control tasks are created when a point-to-point movement is requested, executed for the duration of the movement, and are destroyed once the axes reach their destination. Communication on the Alt-Az is handled in a high-priority task because it must provide axis position information in real time to the MC when called upon.

The MicroC/OS-II kernel is available to the Rabbit developer as a Dynamic C library. In addition to preemptive multitasking and real-time functionality, the kernel provides a means to use dynamically allocated memory, a feature that is missing in Dynamic C. From a systems programming point of view, the style of code required to implement

multiple tasks is similar to Win32 or UNIX threads, and less cumbersome than having to use Dynamic C's other form of multitasking, called cooperative multitasking.

Cooperative multitasking differs from preemptive multitasking in that its tasks, called costates, are not pre-empted by higher-priority tasks, but rather, yield program flow to other costates when it has finished its current activity. The costate's state (being the value of its variables and the identity of the last executed statement) is not automatically saved when it yields control to another costate, so explicit mechanisms, such as state machines, must be written into the code to account for this. State machines can be implemented within a costate by breaking the program flow into discrete steps by means of a switch or if/else statement, where each step represents a certain state in the costate's execution. When the costate comes into context (i.e., is yielded CPU time), it checks its state variable, jumps to the corresponding position in its code, executes to the end of its block of statements, increments its state variable, and yields control. Cooperative multitasking with costates works well in simple programs that perform multiple tasks at different time intervals, and can potentially be more efficient than preemptive multitasking. Due to IRMA's complex program flow, cooperative multitasking was rejected, because the resulting code would have required building very complex state machines.

3.3. IRMA Communication

A significant proportion of the IRMA control software is devoted to handling messaging among the CP, MC and Alt-Az. IRMA uses a binary packet-based messaging protocol that can be used for serial or network communication, which consists of a data payload wrapped with a header describing the packet's identity and payload size, and a footer containing the packet's cyclic redundancy check (CRC) value, which the packet recipient uses to verify the packet's integrity. Timeout timers are used with every communication transaction, whether network or serial based, in order to handle possible communication errors, which if not handled gracefully, could bring the system to deadlock.

The MC acts as a slave of the CP, thus the MC never initiates any communication, but reacts to commands sent by the CP. When the CP sends a command to the MC, the MC first responds by issuing an acknowledgement packet, followed by a packet indicating the start of the requested activity. A data packet is generated if the given command generates information, such as reading the ADC, and a final packet indicates that the requested activity has concluded. Although the command protocol is verbose and consequently slower than transmitting commands and data directly, it is designed to give the operator maximum flexibility in tracing a command when problems arise during communication or execution of the requested activity. Communication between the MC and the Alt-Az controller is performed using a simplified messaging scheme based on the CP – MC protocol, which eliminates the acknowledgement and status packets, but retains the CRC check in order to speed up the transaction time. Since the serial messaging task polls the serial buffer every 100 ms, the round-trip time of the Alt-Az position read command ranges between 20 and 80 ms.

IRMA communication is ultimately dependent upon the instructions provided by the operator. Early in IRMA's development, it was unclear what kinds of activity sequences IRMA was going to perform, so maximum flexibility in commanding the instrument was required. This was achieved by constructing a set of primitive commands represented by a small scripting language with which to individually control IRMA's hardware components. Each statement in the scripting language, called "IRMAscript", is converted by the CP into a binary command packet and sent to the waiting MC, which executes the action associated with the command packet, such as turning on the chop wheel motor or powering up the cryo cooler. By grouping multiple commands into a single script, the operator can specify IRMA's behaviour in a precise and flexible manner. An effort was made to make IRMAscript easy to read and write by the operator, as well as provide mechanisms within the language to permit repetition, conditional flow control, variable assignment, pausing and file I/O. A GUI-based script generation program has been created to expedite and simplify the generation of syntactically correct IRMA control sequences.

It was originally planned that the MC receive, store and interpret incoming IRMA command scripts. In order to expediate IRMA's software development, these functions were offloaded onto the CP, which allowed the interpreter to be written in Perl, a rapid-development scripting language. The CP, besides hosting the IRMAscript interpreter and data archiver, acts as IRMA's user interface, whereby users can upload command scripts for immediate or future execution (using the UNIX scheduler).

4. HARDWARE / SOFTWARE DESIGN CONSIDERATIONS

Software development on the Rabbit-2000 platform using the Dynamic C compiler is in many ways similar to developing programs on desktop PCs. Syntactically, Dynamic C is nearly identical to ANSI C, differing only in the way it defines libraries, as well as slight differences in data types (there are no double-precision floating point numbers, and the default integer size is 16 bits, rather than 32 bits). Because of these similarities, it is not difficult to port 32-bit Windows or Linux code to the Rabbit, being comparable to porting DOS code to Windows. For example, the ADC control code for IRMA, originally written as a Windows dynamic linked library (DLL), required only a day's effort to port it to the Rabbit 2000 processor. The biggest difference between writing software on the PC and the Rabbit is that programs are not compiled on the same hardware that runs the program. Instead, the source code must be cross-compiled on one machine and then transferred to another for execution. The process of transferring the executable is a major obstacle in the Rabbit development process, taking nearly three times longer than the time required to compile the program. The IRMA MC software, being a moderately sized, non-trivial program of 145 kB, requires nearly 30 seconds to load into the Rabbit 2000's flash memory. Taking the program transfer delay into account each time a software change must be made as the program is cumulatively built up and debugged, the length of time to build a program can quickly multiply, making Rabbit software development a lengthy and tedious affair.

Dynamic C is the only high-level language compiler available for the Rabbit-based microcontrollers, and is produced by Rabbit Semiconductor's parent company, Z-World. Available only for Windows, Dynamic C has the look and feel of other GUI-based integrated development environments (IDE), and features a text editor, compiler, plentiful program examples, and source code for all of Dynamic C's libraries. Compiling and executing code is done in the same fashion as on other IDEs, except that compiled data must be transferred into the Rabbit controller before it can be run. If the program is compiled in debug mode, simulated console I/O is available over the Rabbit programming cable, allowing the use of printf statements to write messages, and scanf statements to read keyboard I/O in a Rabbit program. These commands are intended for debugging purposes, and are not available when the Rabbit controller is permanently loaded with the final version of the program.

Code generated by the Dynamic C compiler, although efficient and small in size, is not as efficient as hand-written assembly code. Routines that strobe the Rabbit 2000's DIO lines execute nearly four times faster when written in assembly than code generated by the Dynamic C compiler. Rabbit assembly language routines can strobe DIO at the same rate PCs can strobe lines on their parallel port. The IRMA MC data collection ISR, which samples the IR signal plus one of IRMA's meteorological channels upon receipt of an interrupt signal from the chopper notch, is written in assembly in order to execute with maximum speed. Every moment spent in the ISR prevents all other tasks from executing, so ISRs should terminate as soon as possible. As the Rabbit 2000 microprocessor is largely based on the Zilog Z-80/180 design and shares many of its opcodes, reference and tutorial resources available on the World Wide Web or in books can be applied to Rabbit assembly programming. Dynamic C allows for embedding of assembly code within a C program, or vice versa. From a memory standpoint, assembly language subroutines can only exist in the root (64kB) memory pool, as opposed to C functions, which by default are placed in the extended memory (up to 1 MB).

Network communication plays a central role in IRMA's operations, and was one of the major deciding factors in selecting the Rabbit platform, because of its strong networking support. Because of the Rabbit 2000 CPU's limited processing power, it can only handle data at one quarter the rate of a networked PC. Connected to a traffic-free network using maximum size (1500 byte) Ethernet packets, the Rabbit, doing nothing else, can at best transmit ~270 kB/s and receive ~220 kB/s over the network. Under realistic conditions, where the Rabbit is on a busy network and performing various I/O and computational tasks, its network speed limitation is not a serious problem since a typical Rabbit application cannot generate data at rates that would overload its network transmission limits. IRMA data collection scans typically generate around 500 bytes every 4 seconds.[1]

It is possible to exhaust the non-volatile memory resources on a Rabbit microcontroller module in a short amount of time, if the program writer is not careful. Since the Rabbit is a diskless system and relies on its flash memory chip for permanent storage, including storage space for its executable program, simulated disk file systems can be no larger than the remaining memory. At most, a RAM or flash based file system are limited to 512kB. Care must be taken to write to flash memory efficiently and sparingly, as the flash memory chips used on Rabbit 2000 microcontroller boards are only rated for 10,000 writes. Dynamic C documentation recommends that data be buffered before writing to flash since

writing a large block of data wears the flash unit as much as writing a single data point. The easiest way around this problem is to use a RAM-based file system, which, although it is not permanent, can tolerate unlimited writes. The approach taken by the IRMA control software is to send data over the network to some other networked host. The RAM/flash file system requires much attention to tedious, low-level detail that makes it far more complex to use than disk I/O on a desktop PC. If large capacity, quickly accessible, scratch space is required in an application, using extended memory (xmem) arrays are easy to implement in Dynamic C, and are very efficient.

The IRMA MC and Alt-Az units are both mated to their respective custom-designed circuit boards via the two header rows on the microcontroller unit's reverse side. Although it is possible to use a Rabbit microcontroller module as a stand-alone unit without any buffering of its DIO lines, it is not advisable because they are CMOS based, which makes them susceptible to electrostatic discharge damage. Rabbit microcontroller modules are primarily designed to be plugged into custom-designed circuit boards for projects requiring computer processing and/or network connectivity. For simple control and data acquisition projects, prototyping, or where no electronic fabrication expertise / resources are available, it is easier to use a PC-based DIO card. Rabbit Semiconductor sells a prototyping board to serve these purposes. Due to the longer time required to write Rabbit-based software and build its interfacing electronics, it can potentially cost more to develop a Rabbit-based system than to build a PC-based system. With these caveats in mind, using the Rabbit 2000 in a system may still be worthwhile if it fits into the system's design requirements, and if PC-level performance (speed, storage capacity) and versatility it is not expected.

5. CONCLUSION

Rabbit 2000 microcontrollers can be used effectively in complex instrumentation such as IRMA, which integrates custom-built electronics, mechanical components, and computer power. Through the Dynamic C compiler, the Rabbit software developer has available a rich set of libraries with which to perform networking, serial I/O, a diskless file system, and real-time preemptive multitasking. Although Rabbit microcontroller modules are inexpensive devices, development of control and data acquisition software can cost more than a PC-based project in terms of time and money due to its lengthy development process. In addition, the microcontroller modules need to be integrated with custom electronics and a power supply. Programming the Rabbit 2000 is accompanied by a steep learning curve, as many of its services, such as networking and file system access, are programmed quite differently on the Rabbit 2000 than on desktop PCs. The Rabbit 2000 platform is a minimal, embedded 8-bit controller, not a PC. It would be unreasonable to expect it to perform like a PC. Rather, it should be applied to those specific situations that require computational and communication functionality in the smallest possible space, using the least amount of power. Experience with three IRMA units from field tests at the Smithsonian Millimeter Array (SMA) at Mauna Kea, Hawaii, have confirmed the suitability of the Rabbit 2000 microcontroller family for this application.

6. ACKNOWLEDGEMENTS

The authors would like to thank the following people involved in the development of the IRMA project: B. G. Gom and G. J. Smith, for developing the IRMA concept and instrument design; R. R. Phillips, for work on IRMA's detector and leading the project to completion; G. J. Tompkins, for designing the IRMA electronics. Special thanks to P. Davis and T. R. Fulton. D. A. Naylor acknowledges financial support from NSERC, ASRA and the University of Lethbridge.

7. REFERENCES

1. D. A. Naylor, B. G. Gom, "Remotely Operated Infrared Radiometer for the Measurement of Atmospheric Water Vapor", in Infrared Technology and Applications XXVIII, Proceedings of SPIE, 4820, pp. 908-918, 2003.
2. ALMA Site Characterisation Team, "Atacama Large Millimeter Array Weather Data", http://alma.sc.eso.org/htmls/meteo.html, 2004.
3. Rabbit Semiconductor, "The RCM2100 RabbitCore microprocessor core module", http://www.rabbitsemiconductor.com/products/rcm2100/index.shtml, 2004.
4. Rabbit Semiconductor, "The RCM2000 RabbitCore microprocessor core module", http://www.rabbitsemiconductor.com/products/rcm2000/index.shtml, 2004.
5. Z-World, Inc., Dynamic C TCP/IP User's Manual, p. 59, Z-World, Inc., 2002.
6. Micrium Inc., 949 Crestview Circle, Weston, FL, 33327, USA, www.micrium.com