# SINGLE-CHIP MICROCOMPUTERS EASE PROTOTYPE DEVELOPMENT

*by Simon Taylor,*
*RCS Microsystems Ltd.*

When designing small control systems, or similar items such as high-volume consumer products, much attention is now paid to the possibilities offered by single-chip microcomputer devices.

Often, however, the manufacturers of such devices have a tendency to put off such applications by the necessity of large capital investments for items such as development systems and in-circuit emulators, as well as devices only being available in mask-programmed quantities.

Some single-chip microcomputers for small-volume production or prototyping, can be programmed using any computer that can generate 6502 code, (the BBC microcomputer for example) and a low cost in-circuit emulator system, available for around £2,200.

This allows the designer to specify single-chip devices without the fear of being caught with high development costs, which can prove difficult to recoup from initial sales of a new product.

Most of the single-chip microcomputers described here will replace up to four or five discrete microprocessor parts as well as memory devices and addressing logic. Cost savings are substantial with chip counts greatly reduced, and board space cut to a minimum.

The range is based on the 6502 processor, and has various configurations:

(i) standard microcomputers (6502 + extensive I/O)
(ii) Forth based microcomputers
(iii) intelligent peripheral controllers
(iv) mask programmed versions.

The R6501Q and R6511Q (Fig. 1) offer the R6502 CPU, 32 lines of I/O, 192 bytes of static RAM (32 bytes of which can be battery backed-up on power down), two 16-bit counter/timers (one of which can be used to drive an internal ACIA), ten sources of interrupts and the facility to expand externally to up to 64Kbytes of memory.

The I/O ports and registers are all located in zero page, and therefore make use of the faster cycle times of 6502 zero-page instructions. Four new zero page instructions have also been added: SMB, RMB, BBS and BBR.

● SMB (set memory bit) This allows an individual bit to be set in zero page. For example, SMB 4,02 will set bit 4 in location 2 to a 1 (this is bit 4 on port C). The accumulator is not corrupted, nor are the flags. An equivalent instruction sequence on a standard 6502 is as follows: PHA, PHP, LDA 02, ORA £$10, STA02, PLP, PLA. This sequence uses 24 machine cycles, whereas SMB only uses 5.

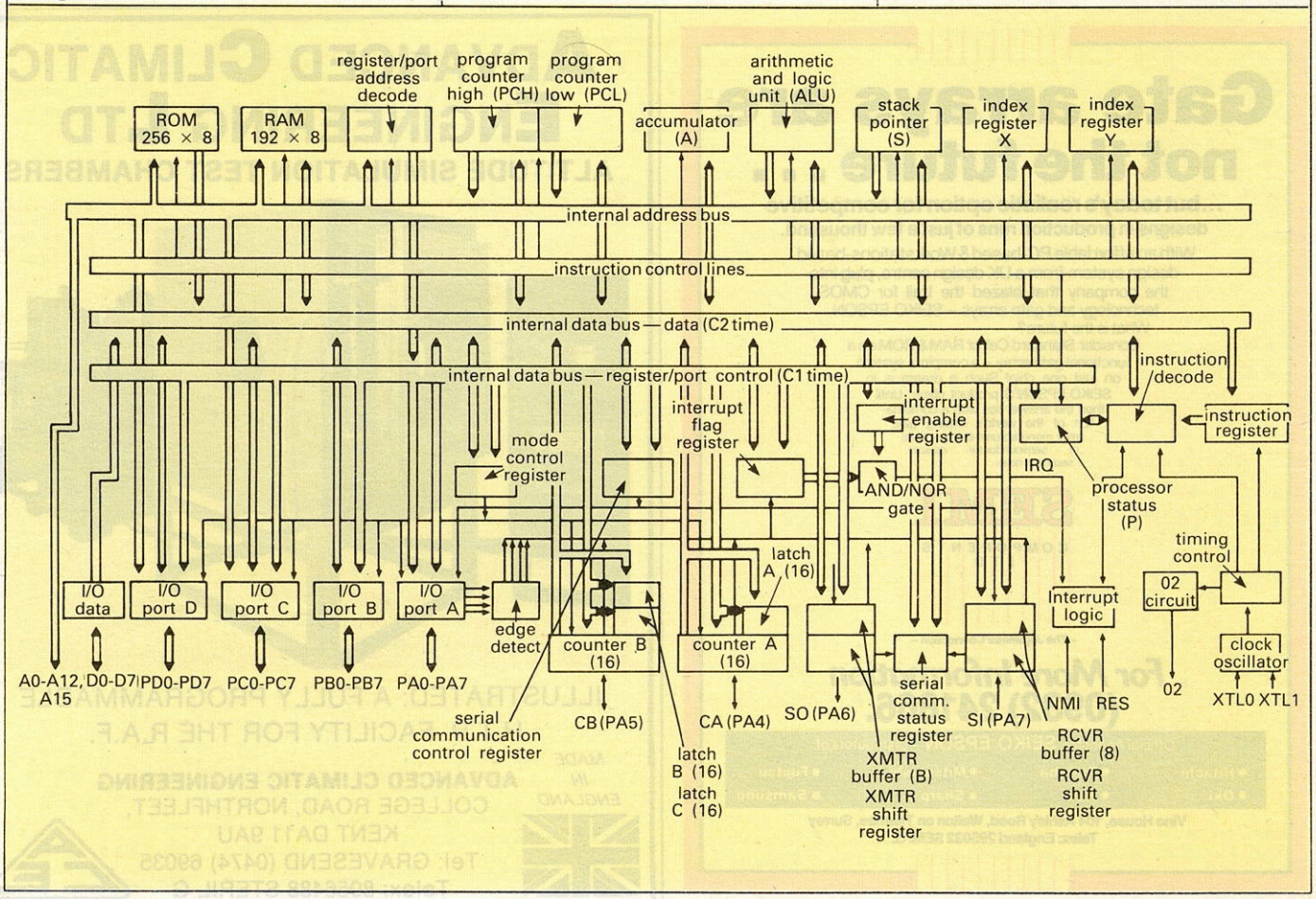


*Fig. 1: R6501Q and R6511Q block diagram.*

● RMB (reset memory bit) This is exactly the same as SMB, but sets the zero page bit to zero.
● BBS (branch on bit set) This instruction is a new form of 6502 instruction in that it has two operands. The first defines the location to be tested (the instruction itself defines which bit), and the second gives the relative offset to jump to. This new instruction gives extra power to these devices in that a bit of I/O, an option switch for example, can be tested and the appropriate action taken in a single instruction without damaging flags or the accumulator.
● BBR (branch on bit reset) The same as BBS, but acts on the bit being low instead of high. This is the instruction more commonly used for switch type inputs as the I/O ports float high when set to input and are generally shorted to zero volts with a switch.

The way that the I/O ports operate is slightly unusual in that there is no data direction register, but the ports are configurable as either inputs or outputs. Consider a standard TTL type input with a PNP type transistor between the input of the gate and ground. To operate the device as an output device, one or zero is written to the base of the transistor. When the data is zero, then the transistor effectively shorts the output to ground.

When the data is one, then the transistor is open circuit and the input of the gate floats high. So, to operate as an input simply write one into the relevant I/O locations and drive the I/O pins from the external devices. The idea is simple and works very well in practice.

The two 16-bit internal timers can be used in a similar way to those on the 6522 device, with timer A also finding use as a baud rate generator for the serial input and output channel. The counters can both be used as follows:

counter A – pulse width measurement
– pulse generation
– interval timer
– event counter

counter B – retriggerable interval counter
– asymmetrical pulse generation
– interval timer
– event counter

Counter A can also be used as a baud rate generator for the internal serial I/O channel. This can be configured in many different ways to give between 5 and 8 bits of data with odd or even parity and 1 or 2 stop bits. Data rates of up to 4800baud can be generated with a 1MHz clock, but if a 1.8432MHz crystal is used, giving a 916kHz clock, then baud rates up to 19,200 baud are feasible with no data rate errors.

Interrupts can be generated from counters attaining pre-set values, serial registers being full or empty, or from edge-detect inputs on port A. These edge detect inputs are useful for functions such as acknowledge signals on Centronics type interfaces, or "conversion complete" signals on ADCs for example. The R6501Q and R6511Q are identical except on two points: (i) the R6501Q has a divide-by-four clock, where the R6511Q has a divide-by-two clock, and (ii) the R6501Q has pull-up resistors on its I/O ports and the R6511Q has open collector outputs.

Two single-chip Forth based microcomputers, the R6F11 and R65F12 (Fig. 2), based on the R6501Q, add a 3K Forth kernel to the device. This 3K contains all of the fundamental workings of Forth to enable the user to add up to 16K of program written in Forth to configure a target application. For development, one simply adds a Forth development ROM which adds compiling words and words to enable the generation of compact, headerless code. This makes the use of a high-level language very competitive in terms of memory to an equivalent machine-code program.

Code is written interactively using the development ROM with a terminal connected, enabling development and testing of the code inside the target system itself. When the code is complete, then a ROM is programmed, and the code re-inserted with auto-start parameters so that the program is executed immediately on power-up.

The intelligent peripheral controller is probably best suited to the area of microcomputer add-ons. The device is again based on the R6501Q, and has one of its ports configured as two memory locations in a host computer's address space. Ten I/O bits are used, eight for connection to the host computer's data bus, with an address bit (0 or 1) and a read/write line. There is a register available in the memory map of the R6541Q which is used as a data in and a data out register for data communicated

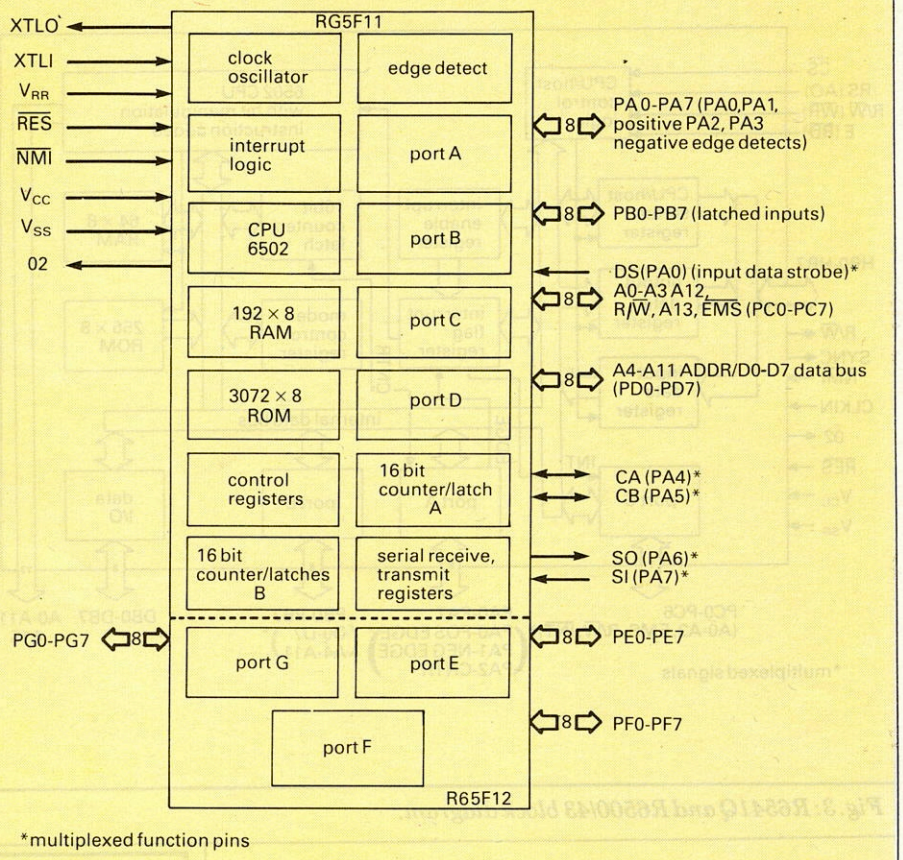


*Fig. 2: R65F11 and R65F12 interface diagram.*

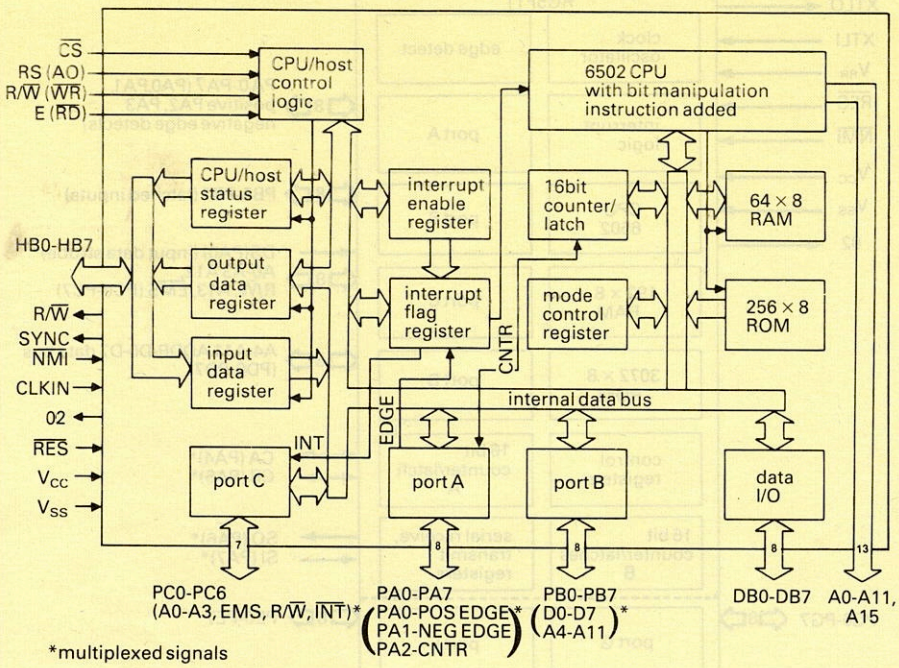


*Fig. 3: R6541Q and R6500/43 block diagram.*

through this port and into the data bus of the host system (Fig. 3).

A flag is provided in the interrupt flag register of the R6541Q which signifies that new data has been written into the host bus data register.

This device could find applications as an I/O controller on a home computer for example, as there is plenty of I/O available on the device for use as Centronics interfaces or such like. The device would only take two memory locations from the host, and could provide data buffers, disk control enabling the host to get on with the business of computing.

Most of the devices discussed here have a mask-programmed equivalent for high-volume, low-cost production. Mask-programmed devices need to be ordered in minimum quantities of one thousand, and can sometimes be supplied in as short a time as six weeks from the time of providing program code in EPROM. Mask-programmed devices offer cost savings in production quantities even as low as three hundred over their "prototyping" or low-quantity equivalents.□

*Write in no.* **402**