

University of Waterloo
Faculty of Engineering

Design and Implementation of a Bit Error Rate
Tester

SilCom Research Limited
Kanata, Ontario

Gavin J. Hurlbut
92012471
3A Electrical Engineering

Originally written September 18, 1995

September 11, 2002

Contents

Recommendations	v
Conclusions	vi
Summary	vii
Contribution	1
1 Background	2
1.1 Company Profile	2
1.2 POCSAG paging protocol	2
1.2.1 Synchronization Codeword	3
1.2.2 Address Codeword	3
1.2.3 Message Codeword	3
1.2.4 Idle Codeword	6
1.2.5 Error Detection and Correction	6
1.3 Bit Error Rate vs. Successful POCSAG message	7
1.4 Bit Error Rate Detection methods	7
2 Design Requirements	9
3 Design Choices	10
3.1 Assumptions Made	11
4 Prototyping and Debugging	12
5 Software Design	13
5.1 Expandability	15
5.2 Memory Allocation	15
5.3 Serial Interface Protocol	17
5.3.1 Flow Control	19
5.3.2 Data Protocol	19
5.3.3 Implementation	22
5.4 Module Contents	23

6	Hardware Design	25
6.1	Expandability	25
7	Realizing the Hardware	26
8	Testing of the Design	27
8.1	Testing Sync-Sample-Compare	28
8.2	Testing Gated Mode	28
8.3	Testing Serial Protocol	29
A	Glossary	31
B	Equations	32
C	Schematic	33
	Bibliography	35

List of Figures

1.1	POCSAG Protocol — Synchronization Codeword	4
1.2	POCSAG Protocol — Address Codeword	5
1.3	POCSAG Protocol — Message Codeword	5
1.4	POCSAG Protocol — Idle Codeword	6
1.5	Probability of n bit errors vs. Bit Error Rate – Two Codewords .	8
5.1	Problems with SYNC and DELAYED SYNC timing	14
5.2	EEPROM Jump Table Algorithm	16
5.3	Memory Map	18
5.4	Serial Protocol — General Packet Structure	19
5.5	Serial Protocol — Command Packet Structure	20
5.6	Serial Protocol — Bit Logging Array Element	21
5.7	Serial Protocol — Event Logging Packet Contents	21
5.8	Serial Protocol — Read Memory Packet Contents	21
5.9	Serial Protocol — Write Memory Packet Contents	22
5.10	Serial Protocol — Memory Response Packet Contents	22
5.11	Serial Protocol — Execute Packet Contents	22
5.12	Serial Protocol — Buffer Structure	23
C.1	Bit Error Rate Tester Schematic	34

List of Tables

5.1	Serial Protocol Packet Types	20
5.2	Serial Protocol Control Commands	20
5.3	Source Files	24

Recommendations

Conclusions

Summary

Contribution

Chapter 1

Background

1.1 Company Profile

Silcom Research Limited (SRL) is a product development company specialising in research and development of world-class radio communications products. Their products have included advanced paging products, including a PCMCIA product and two-way data products for fixed point radio communications.

1.2 POCSAG paging protocol

The POCSAG paging protocol is the standard radio paging protocol used throughout the world. It was developed by the British Post Office (now British Telecom) between the years of 1978 and 1980 and was accepted as an international standard by the International Radio Consultative Committee (CCIR) of the International Telecommunication Union (ITU) in 1986. Up until that point, several different paging codes were being used. These other codes used differing error detection/correction codes and differing data structures, but were all similar to POCSAG in many ways.

The POCSAG protocol specifies code format along with battery economy guidelines and modulation schemes and other specifics. For the purpose of this report, only the code format is of interest.

POCSAG paging data is sent in units known as batches. Each batch contains 17 codewords – one synchronization codeword plus 8 frames of 2 codewords each. Each of the codewords is 32 bits long, and have differing structures depending on the type.

Each pager is assigned a (usually) unique 21 bit address. If two pagers have the same address, they will both receive all pages to that address. The least significant 3 bits of the address of a pager determines which frame of a code batch the pages for that pager is received in. This is done to allow pagers to utilize battery saving techniques (such as only powering up their receivers in a

given frame). The remaining 18 bits are sent in an address codeword which is sent at the beginning of the appropriate frame.

The beginning of a POCSAG transmission contains at least 576 bits (one batch plus one codeword) of preamble – alternating ones and zeros. This allows a pager receiver to wake up at any given frame in a codeword and synchronize to the preamble bits before decoding the following data.

The end of preamble is marked by the beginning of the synchronization codeword. As soon as the pager detects the synchronization codeword, it can then sleep until the frame where its data should appear. If there are unrecoverable errors in the sync codeword, the pager will abort receiving and sleep for the amount of time that corresponds to a batch length to save battery life.

In any given frame, the first codeword will usually be an address codeword. A long message to another address can spill over from one frame into subsequent frames, so if the first codeword is *not* an address codeword, there is no message in that batch for the pager which can then sleep until the end of the batch. If the address codeword matches the pager’s address, the next codeword is the message for the pager, and subsequent codewords until another address or an idle codeword is found, at which point the message is finished.

1.2.1 Synchronization Codeword

The synchronization codeword is shown in Figure 1.1. Note that it is also a valid address codeword, so the contained address must not be assigned to any pager.

1.2.2 Address Codeword

The structure of an address codeword is shown in Figure 1.2. There are two addresses that must not be assigned to a pager. They are the ones that correspond to the sync codeword and the idle codeword.

The first bit of an address codeword is always a binary “0”. This distinguishes it from a message codeword as quickly as possible. Following the flag bit is 20 bits of data followed by 10 bits of check bits (which use a 31:21 BCH (Bose–Chaudhuri–Hocquenqhem) code — See Section 1.1.5), and one parity bit (to obtain even parity).

In an address codeword, the 20 bits of data contains the 18 most significant bits of the pager’s address followed by a two bit function code (which is to notify the pager of what type of message will follow).

1.2.3 Message Codeword

The message codeword (see Figure 1.3) follows the same basic format as the address codeword . The flag bit for a message codeword is a binary “1” to distinguish from address codewords. The 20 bits of data take different formats. The most common are 4-bit BCD (Binary Coded Decimal) which is used for numeric paging, and 7-bit ASCII which is used for alphanumeric paging. The

Bit No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit	0	1	1	1	1	1	0	0	1	1	0	1	0	0	1	0

Bit No.	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Bit	0	0	0	1	0	1	0	1	1	1	0	1	1	0	0	0

Figure 1.1: POCSAG Protocol — Synchronization Codeword

Figure 1.2: POCSAG Protocol — Address Codeword

Bit No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Contents	0	Address Bits														

Bit No.	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Contents	Address Bits			Function Bits		Check Bits										Even Parity

Figure 1.3: POCSAG Protocol — Message Codeword

Bit No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Contents	1	Message Bits														

Bit No.	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Contents	Message Bits					Check Bits										Even Parity

Figure 1.4: POCSAG Protocol — Idle Codeword

Bit No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit	0	1	1	1	1	0	1	0	1	0	0	0	1	0	0	1
Bit No.	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Bit	1	1	0	0	0	0	0	1	1	0	0	1	0	1	1	1

7-bit ASCII data spans message codewords (as 20 is not evenly divisible by 7). Thus the third character of an alphanumeric message will have six bits in one codeword and the final bit will be the first bit in the next message codeword's data.

The check bits and parity bit follow the same structure as in an address codeword — 10 bits of 31:21 BCH check bits and one even parity bit.

1.2.4 Idle Codeword

The contents of an idle codeword are shown in Figure 1.4. The idle codeword is sent when there is no more data to follow, and there is not another address codeword to send in the given batch. The arrival of either an idle or address codeword is what signifies the end of message data to a receiving pager.

Again, note that the idle codeword is a special case of the address codeword (it is valid as an address codeword), so the address bits that are contained must never be assigned to a pager.

1.2.5 Error Detection and Correction

The POCSAG paging protocol uses a 31:21 BCH code. The flag bit and the 20 data bits correspond to the coefficients on a polynomial with terms from x^{30} down to x^{10} . This polynomial is then divided (with all coefficients being modulo-2) by the generating function

$$g(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1 \quad (1.1)$$

The check bits then correspond to the coefficients of terms in the remainder that are from x^9 down to x^0 .

The check bits in the address and message codewords allow for not only error detection, but also for two bits of error *correction* per codeword. The parity bit will allow for aiding in ensuring that the check bits themselves are received correctly.

A 31:21 BCH code such as the one used by the POCSAG paging protocol is simple to implement with logic gates, and provides error correction which allows the pager to decode signals with lower signal quality than would be possible with no error correction. This translates directly into better sensitivity for the pager.

Thus for a codeword to be decoded correctly by a pager, there can be no more than 2 errors in each codeword. This will be used as a basis for further relationships between successful decoding of messages and BER (Bit Error Rate).

1.3 Bit Error Rate vs. Successful POCSAG message

For a POCSAG message to be successful, there can be a maximum of two bit errors in each of the first two codewords of the message (one address codeword and one data codeword). If there are three or more bit errors in either of these codewords (or any following message codeword), the pager will abort receiving the message as without very complicated algorithms, the errors are uncorrectable and would cause a message that is likely to be undecipherable by the user. The purpose of the Bit Error Rate Tester is aid in determining the sensitivity of a pager's radio receiver to low level signals. To do this, a relationship between Bit Error Rate (BER) and the probability of having a given number of bit errors in two codewords must be established.

As the BER is simply a probability that a given bit is in error (when expressed in a per-unit manner), the relationship between BER and the probability of having a given number of errors in two codewords (independent of each other) is a simple matter of probability equations. The relationships are shown in Appendix B, and Figure 1.5 show the results graphically.

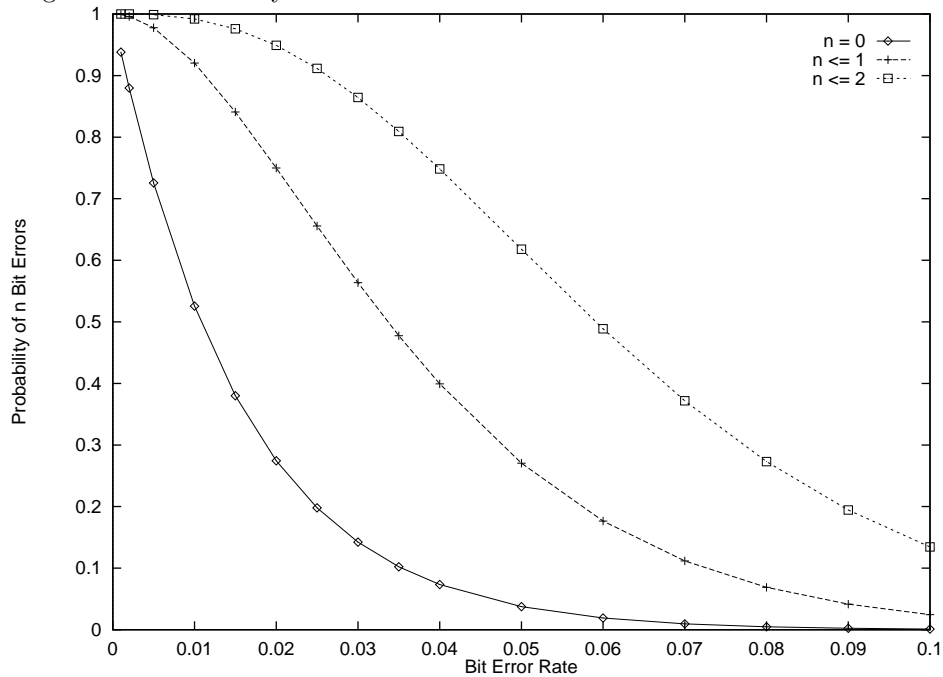
Thus, for a pager to receive 80% of pages with 2 or fewer bit errors in the first two codewords allows for a channel with a maximum bit error of about 0.035 (or 3.5%) as interpolated from Figure 1.5. Solving equation B.7 for $P_2(n \leq 2) = 0.80$ gives a result of $p = 0.0357919$ (or 3.57919%), so the estimation made from the graph is a decent one.

1.4 Bit Error Rate Detection methods

There are several methods of performing a bit error rate detection.

One method is to perform a logical exclusive-OR between the demodulated output of the radio and the modulation input to the transmitter. This will give an output of a signal that is a logic low whenever the demodulated signal is not in error, and a logic high whenever it is in error. When using TTL logic (or CMOS) the logic low is ground (0V), so to calculate an error rate, simply taking an integration of this pulse stream will suffice. The ratio of the integration of the pulse stream to the integration of a constant logic high (over an equal amount of time) will then give the error rate. This method requires an XOR gate and a method of integration.

The next method requires more hardware to implement. It is a method that requires synchronization to the edges of the known good data. The algorithm involves sampling multiple times in the middle of each bit, and deciding the level

Figure 1.5: Probability of n bit errors vs. Bit Error Rate – Two Codewords

of the bit (logic “1” or logic “0”) by a majority vote of the samples. This level is then compared to the expected value (as determined by the level of the known good data during the same bit). This method requires either a microprocessor or a fairly complex ASIC (Application-Specific IC) to implement.

The final method to be mentioned here is one that implements the BERT as a ratio dividing counter. The modulation-in signal (the known good data) would be applied as signal A, and the demodulated radio data would be applied as signal B. The output of this ratio counter would be $\frac{A}{B}$ (or $\frac{B}{A}$), which then could be correlated to the BER. The principle behind this method is that when the demodulated signal has little error, the ratio of the frequencies of A and B would be 1.000 as there would be an equal number of edges in both signals. As the signal level decreases (and bit error increases), an increasing number of false edges would appear in the demodulated signal. Thus the ratio $\frac{A}{B}$ would decrease as the rate of B would be rising. The difficulty of this method is the correlation of the ratio $\frac{A}{B}$ to BER is not an intuitive task. An additional problem with such a design is that if the demodulated signal has an error rate above 50%, it would be reported as $100\% - BER$ as the ratio would be identical (just a phase difference).

Chapter 2

Design Requirements

The Bit Error Rate Tester (which will be referred to as the BERT henceforth) is needed as a test instrument to determine the sensitivity of radio pagers. Given a known good signal (the one that is presented to the transmitter), and the digital output of the pager's radio, the BERT's task is to determine the rate of error on the demodulated signal.

At SRL, the BERT was required to perform more than simply report an error rate on a continuous basis. The requirements also included a user interface that would be appropriate for a lab environment (including an analog indication of error rate), a facility to log the accumulated data on a host PC when required, and a method of accumulating separate error rates for when a gating signal is present and when it is not, along with the overall error rate. It also was to include a user-adjustable delay in both the known good data (to compensate for propagation delays in the transmitter path and in the radio) and in the gating signal (to accommodate radio turn-on time delays).

The bit rate of the data presented to the BERT was to be adjustable between just below 512 baud and just over 2400 baud to accommodate the common paging data rates of 512, 1200 and 2400 baud, and to allow testing of the pager's sensitivity to bit rates that are not quite correct.

The design of the BERT was to be completed in a way such that any future development would not require an extensive overhaul to implement. This was necessary as the primary design was done by a student on a work-term, and any future development would be completed by either a different co-op student or by a full-time employee, but almost certainly not by the original designer.

Chapter 3

Design Choices

The choice was made to implement the BERT using a microcontroller to do the majority of the work. This allows a fair amount of flexibility in the design if it were to be modified at a later date. Additional factors in choosing a microcontroller-based system were the ease of development (as compared to using an ASIC or discrete logic), the small number of IC packages (and hence smaller board size) that such a design would allow, and the general level of comfort of the participants in the project.

The microcontroller of choice was Motorola's MC68HC11K4 an eight bit microcontroller with built in ROM, RAM, and EEPROM, and multiple ports (both parallel and serial) with several special-function ports, running with a 16 MHz clock. As the BERT would require accurate timing measurements to properly function, the timing capture functions of the HC11 made it a good choice. The choice of the K4 version of the 68HC11 was dictated by the need for a pulse-width modulation function (which it has built in), and by the availability of the processor at SRL.

The user interface was chosen to be an LCD module (two lines of 16 characters) along with pushbuttons to be used to increment, decrement, store values of user parameters and to cycle through the various parameters. An analog meter was also added to display the error rate (as was dictated by the design requirements).

As the BERT was being implemented with a microcontroller, the method of error detection was chosen to be a sync-sample-and-compare method whereby the microcontroller detects edges on the known good data, delays for the length of time the user has selected, and then samples the radio data repeatedly within the width of a bit. The value of the radio data is then determined by a majority of samples. This method of error detection is made possible by the timing subsystem of the 68HC11K4.

3.1 Assumptions Made

To simplify the design of the BERT, several assumptions were made.

To reduce the complexity of the syncing mechanism on the known good data, it was assumed that this data would be a clean square wave with a 50% duty cycle (at TTL levels), and that it would be an audio signal (i.e. for a 512 baud signal, a 256 Hz square wave signal) rather than POCSAG data. This allows the software to assume that the current bit will be of the same duration as the previous bit (which is not at all true of POCSAG data). As the radio will have the same sensitivity to audio signals as to POCSAG data (at a given data rate), the error rate will be comparable to what will appear on real data. At low received signal levels (typically in the range of 100 to -110 *dBm* when doing sensitivity tests), the majority of errors are caused by environmental noise. To allow the use of the BERT with real POCSAG data, the SYNC signal would be used as a serial clock for data, and the POCSAG data would be fed in separately.

To allow an interrupt-based (internal to the microcontroller) sync-and-sample mechanism, it was assumed that at no time would a user require the BERT to operate at data rates beyond the design requirements (just above 2400 baud). This was necessary as the time required to sync on the edges and by the sampling code is too long to allow higher bit rates. If higher bit rates were to be required, either some external logic would have to do the syncing, or a faster microcontroller would have to be used. The overhead of the routines was estimated originally to allow a maximum data rate of about 2600 baud, and the estimate proved to be fairly accurate.

Chapter 4

Prototyping and Debugging

A necessary part of any project is the act of prototyping the system before making the final product. This is especially important when the system gets increasingly complex.

The prototyping of the BERT was done using wire-wrapped circuitry and an evaluation system (68HC11K4 EVM) made by Motorola. The latter allows a developer to use a monitor system to debug HC11 code by setting breakpoints and tracing through code. It uses a serial cable to allow the developer to control it from a PC using terminal emulation software. The EVM allows the developer to place what would normally be in the one-time-programmable ROM of the 68HC11K4 into RAM on the EVM which is designed to emulate the ROM. This facilitates code testing and development as there is no need to program a microcontroller with code to test it. Patching the code in the EVM is also possible, whereas that would be nearly impossible if simply programming a micro.

The software was written in assembly language and was compiled using a macro assembler from 2500AD Software Inc on an IBM PC compatible computer. It was written as relocatable linkable modules so that it could easily be changed at a later date. The output of the linker is a Motorola “S19” format file which the 68HC11K4 EVM is able to download.

After the circuit boards arrived, the circuit was built up and tested along the way. Once the integrity of the boards was confirmed, the wire-wrapped prototypes were discarded and all further testing was performed using the actual PC board (which has the appropriate connectors to allow the use of the EVM instead of using a real processor to allow for further debugging).

Chapter 5

Software Design

The software for the microcontroller in the BERT was designed in a modular fashion. This made it easier to move routines around in memory and to add more routines when necessary.

The majority of the BERT's software operates in an interrupt mode. This makes it considerably more difficult to debug and test as much of the algorithms are dependant on accurate time measurements, and placing breakpoints in the middle of the interrupt handlers destroys this timing.

The mainline (non-interrupt-based) code follows a simple loop. In this loop, the input switches are read, and the output (on the LCD module) is updated. There are delay counters included such that the LCD screen has time to finish updating before more data is sent to it. This eliminates the jitter that the LCD screen exhibits when it is being constantly updated.

To synchronize to the edges of the serial clock input (known as SYNC henceforth) use was made of the timing subsystem of the HC11K4. In particular, the SYNC signal was applied to one of the input capture pins on the micro, which was configured to capture the time of both rising and falling edges of the input signal. The time between successive edges is then the width of a bit. This requires a 50% duty cycle on the SYNC waveform or else the timing of the sampling will go awry.

Once the time of the sync edge has been determined accurately, the user-selected phase-delay is applied to the SYNC waveform to create a DELAYED SYNC waveform that will be used for determining when to take samples of the demodulated radio signal input (known as POCO). The DELAYED SYNC waveform is also supplied as an output so that the user can determine the optimum phase-delay value to use (such that bit error rate is minimized). To implement the phase-delay, the timing subsystem was again used. An output compare timer is set up on every edge such that there is an interrupt generated after the amount of time dictated by the selected phase-delay. For example, at a bit rate of 1200 bits per second and a phase-delay of 90° , the width of each bit is $833\mu s$ and the time-delay used to set the timer would be $417\mu s$ or 1667 clock cycles.

When the DELAYED SYNC timer has expired, the width of a bit (as measured

Figure 5.1: Problems with SYNC and DELAYED SYNC timing

for the previous bit) is divided into 16 parts. This amount of time is termed the sample time (as it is the time between samples taken on the POCO input). A total of 8 samples are taken in the middle of the bit. Therefore, a timer is set for the first sample — which is 4 sample widths (or $\frac{1}{4}$ of a bit) past the DELAYED SYNC edge.

When the sample timer has expired, the timer is set to the next sample, or simply shut off if all of the samples have been taken already. The POCO input is then sampled once. When all of the samples have been taken, the outcome is decided by a majority of samples. The sampled value of POCO (as decided by the vote) is then compared to the value of the known good data within the given bit. A bit counter is then advanced, and if the bit is in error, an error counter is advanced also.

This sync-sample-compare algorithm continues until a user-defined number of bits has been reached (known as the sample window length). When this happens, the error rate over the sample window is calculated. This is placed in a table of error rates which contains the previous seven error rates along with the current one. The average of this eight rate table is then output both to the LCD module, and to the analog meter (to a maximum of 5% – higher error rates will register as the maximum).

As there is the possibility of several interrupts occurring simultaneously, it is necessary to ensure that all of the interrupts are serviced in an efficient manner. As the interrupts in the HC11K4 can be either level-triggered or edge-triggered, the choice was made to use the level-triggered mode to minimize missing interrupts. In addition to this, at the end of each interrupt routine, the status of the timers is checked. If an interrupt was lost or was about to be triggered, the interrupt would be serviced immediately. This removes the possibility of missing a DELAYED SYNC timer that would correspond with a SYNC edge, or of missing a sample that correspond with SYNC edge.

To allow for SYNC signals with imperfect 50% duty cycles (an error of $1\mu s$ can cause loss of synchronization under certain circumstances), an extra check was placed in the DELAYED SYNC algorithm. The problems occur when the phase-delay is very close to 180° as the DELAYED SYNC edge occurs at the same time as the SYNC edge (see Figure 5.1). The double-buffering algorithm will be unstable if the width of each bit is not identical. To deal with this, whenever the bit length is within a small margin of the phase delay (which only occurs near 180°), double-buffering is forced to be active. This averts the problem by enforcing the double-buffering which otherwise would have been shut off every time a bit was longer than was expected.

5.1 Expandability

One of the major design requirements for the BERT was that it had to be easily expandable at a future date. This was especially important as the initial requirements were rather sketchy, and developed as time went by.

To allow future expandability without wasting a microcontroller, a jump table is maintained in the internal EEPROM. This allows future programmers to replace a routine without having to reprogram a new micro. The new routine would simply have to be placed in an empty block of the internal ROM, and the address of the routine would be changed in the EEPROM jump table. All accesses to these routines thus should be through the jump table otherwise the table becomes useless. Jumping to a subroutine via this jump table is slower than accessing the routine directly, (it takes 89 CPU cycles extra – or $22.25\mu s$ at a clock speed of 16 MHz), but the additional modularity makes it worthwhile. However, some of the routines must be as fast as possible (such as syncing to the edges of the data), so any subroutine calls in these routines are done directly. An effort was made to minimize the quantity of external calls in these routines.

The algorithm used to implement the EEPROM jump table is rather convoluted. Figure 5.2 illustrates the algorithm, which is described here. The EEPROM jump table itself is a list of addresses of routines to be accessed. When a routine is to be accessed, the index of the required routine is loaded into an internal register (after saving the previous contents on the stack), and the EEPROM jump table call routine (`ee_jsr`) is called. This routine then carefully stores all the register contents (save the one already preserved) on the stack, and indexes the jump table with the given index. It then pushes the address of the EEPROM jump table routine (`ee_rts`) onto the stack, then the address of the required routine. The registers are then restored to their original values from the stack.

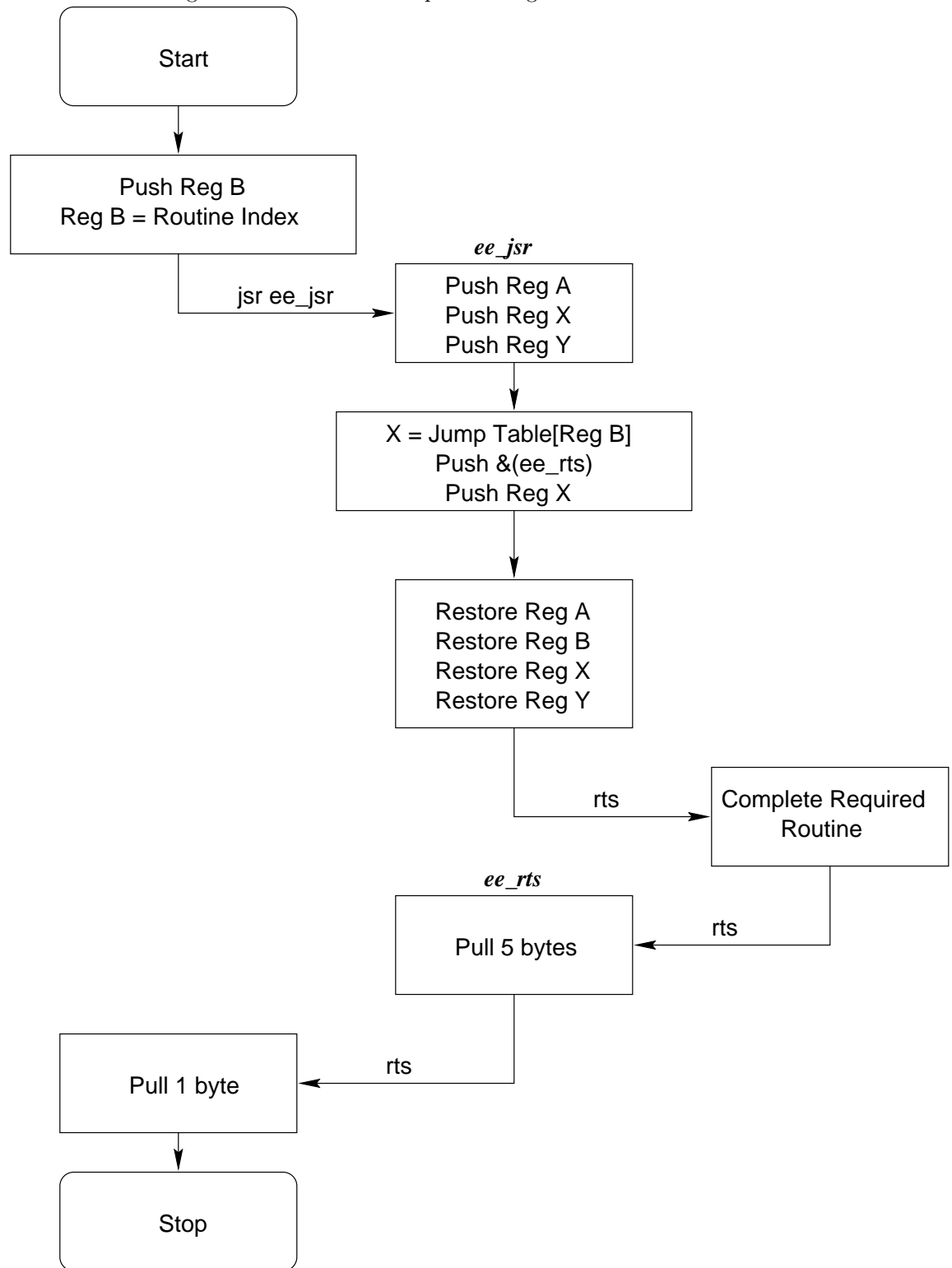
At this point, everything is set up to call the routine. To do so, `ee_jsr` actually does an RTS (return from subroutine). This causes the address of the required routine to be pulled off the stack, and used as the program counter (as if it were the return address of a calling routine). When the routine is done, it also performs an RTS (which is the normal exit from a subroutine). This causes the address of `ee_rts` to be pulled off the stack, and loaded into the program counter. The `ee_rts` routine then restores the stack to the state it was in before calling `ee_jsr`. The return from this point brings the code back to the calling routine, which then removes the last register from the stack.

5.2 Memory Allocation

The MC68HC11K4 has 24K of internal ROM, 640 bytes of internal EEPROM and 768 bytes of internal RAM. The Bit Error Rate Tester is small enough to fit into these internal memories with plenty of room to spare. Figure 5.3 shows the layout of the memory as allocated in the BERT.

The RAM section (from \$0080 to \$0380) is split into two segments in the

Figure 5.2: EEPROM Jump Table Algorithm



code. The first segment (known as PAGE0) extends from \$0080 to \$00FF. The second segment (known as DATA) extends from \$0100 to \$0380. This was done to allow optimizations in the code, allowing such operations as `bclr` and `bset` which require direct addressing. On the MC68HC11K4, direct addressing requires that the upper 8 bits of the address be \$00. The data placed in PAGE0 is mostly bitmapped status bytes.

The stack grows from \$037F towards the beginning of memory. Thus, the end of the DATA segment must be kept near \$02DF or thereabouts to allow enough space for the stack. The stack rarely uses more space than about 150 bytes, so 160 bytes should be enough space to allocate to it. This leaves the size of the PAGE0 segment as 128 bytes, and of the DATA segment 480 bytes.

The EEPROM is split into two segments as well. The first, known as the Fixed EEPROM segment is from \$0D80 to \$0F1F, and is meant for items that are not likely to change in size (with the exception of the EEPROM jump table which is at the end of this segment). The second is the Variable EEPROM segment (from \$0F20 to \$0FFF) which is meant for default text strings and other items that are referenced in tables that are stored in the Fixed EEPROM segment.

The EEPROM jump table is located at the end of the Fixed EEPROM segment so it can grow at a later date. It does not really fit in (logically) with the contents of the Variable EEPROM segment, so the choice was made to fit it in the Fixed EEPROM segment. Ideally it should be contained in its own segment which would reside between the Fixed and Variable EEPROM segments so there is still space for growth. The Variable EEPROM segment also requires space for growth, so the jump table would restrict its growth if placed after it. The Fixed EEPROM segment can not support growth (without massive code changes), so the jump table is best placed directly after it (where it currently resides).

Finally, the ROM has two sections, the CODE segment, and the interrupt vectors. The interrupt vectors of the MC68HC11K4 occupy the last 64 bytes of the ROM – from \$FFC0 to \$FFFF. There are 32 interrupt vectors (each of 16 bit length). The CODE segment occupies the remainder of the ROM – from \$A000 to \$FFBF. This allows for 24,512 bytes of code (nearly 24K). As the code of the BERT is only around 4K, there is plenty of room for expansion at a later date. The code in the BERT starts at the beginning of the CODE segment, so any future expansion would be tacked on after the present code.

5.3 Serial Interface Protocol

The serial port on the BERT is for two purposes: logging events on an external PC, and modifying and probing memory on the BERT. It is implemented with a four line (not including ground) serial cable. The supported RS232 lines are: Tx, Rx, RTS, CTS. The cable is a virtual null-modem cable (in that the Tx and Rx lines and RTS and CTS are crossed — but on the BERT's circuit board, not in the cable). The baud rate of the BERT's serial port is set at 19200, although

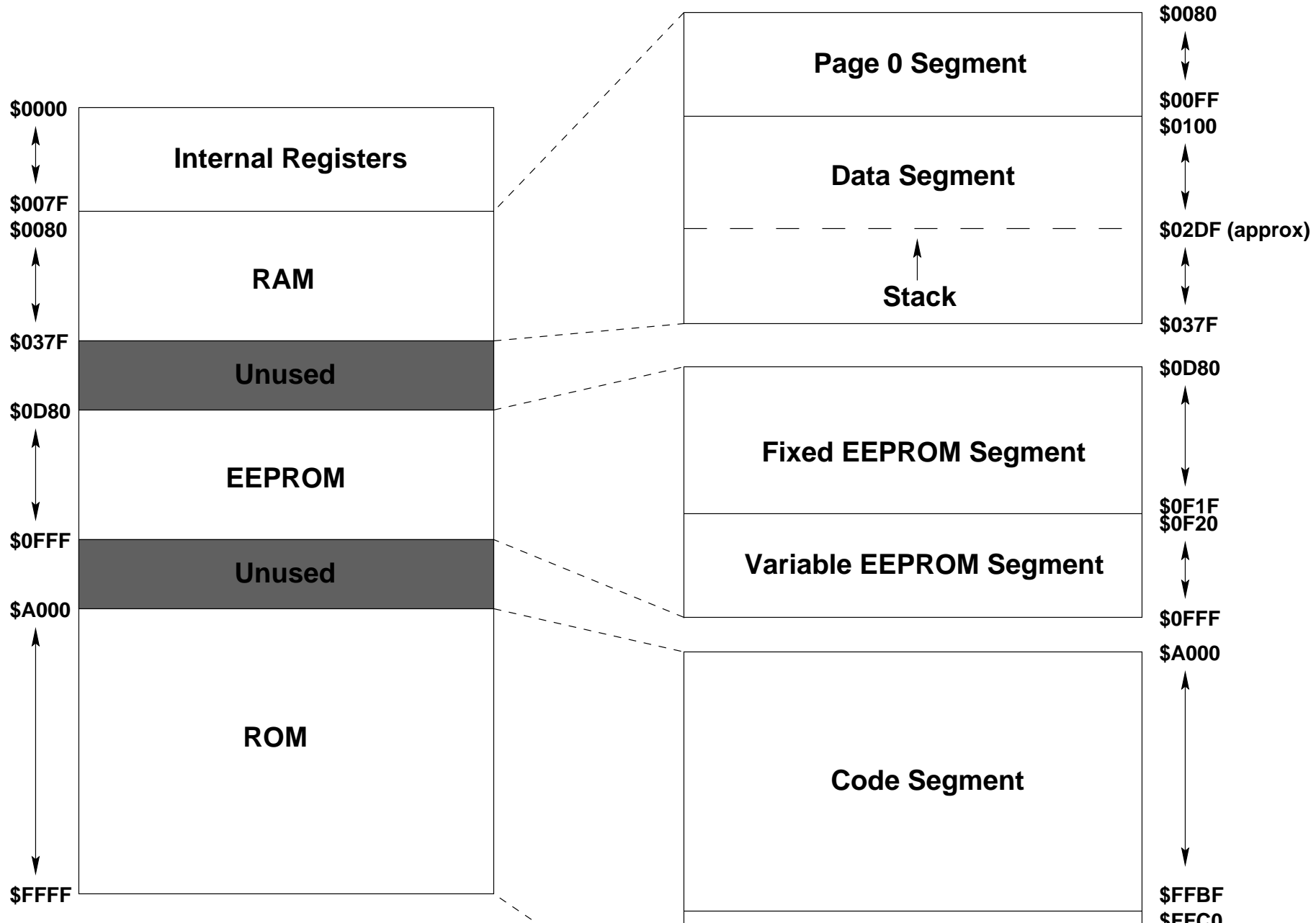
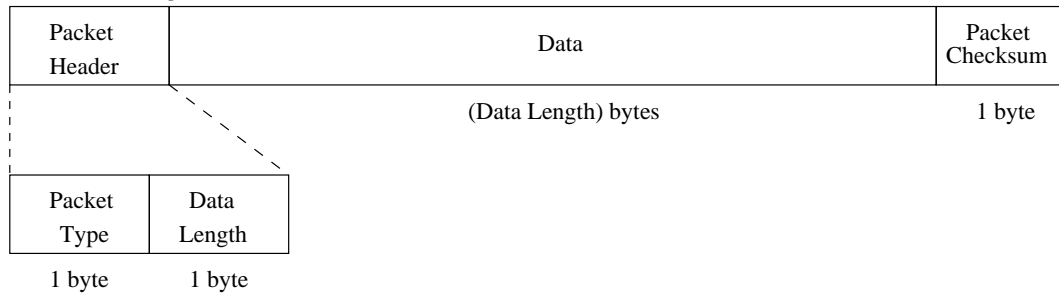


Figure 5.3: Memory Map

Figure 5.4: Serial Protocol — General Packet Structure



it may need to be lowered to 9600 depending on how well the BERT can handle it while not losing sync.

5.3.1 Flow Control

The BERT/PC interface uses a symmetrical hardware flow control. When the BERT is not ready to send (or receive) data, it will negate RTS. When the host PC is not ready, it will negate RTS (which is seen as CTS by the BERT). This will allow for some level of flow control, although hopefully, it will not be needed. This will allow for a slow PC to stop the BERT from overrunning it.

5.3.2 Data Protocol

The data flowing over the link will be 19200 baud, 8 bit, no parity, 1 stop bit to allow compatibility with a vast majority of computer hardware. To start with, the data protocol will be a simple Command/Response protocol with the PC sending all of the commands. Eventually, this will be expanded to be a SLIP-like protocol whereas the BERT could send Event Logging data without being specifically asked to do so.

The data packets have a definite structure to them (Fig. 5.4) The packet header contains the type of packet that follows, and the number of bytes of data (not including the checksum) that will follow. The packets are variable length to a maximum of 19 bytes of data (total length of 22 bytes including the header and checksum).

The checksum is an eight bit checksum biased by \$69 (to reduce the number of \$00 checksums).

A list of packet types is shown in Table 5.1.

All packet types conform to the data structure in Figure 5.4 except Type \$00 (Control) which have no data byte count or checksum, just the packet type (\$00) followed by the singular control character. Thus a control packet takes the form shown in Figure 5.5. A list of these control commands is shown in table 5.2.

Table 5.1: Serial Protocol Packet Types

Type Number	Direction	Contents
\$00	Tx,Rx	Control (ACK, NAK, etc.)
\$01	Tx	Event Logging
\$02	Tx	Sample Window Logging
\$03	Rx	Read Memory Command
\$04	Rx	Write Memory Command
\$05	Tx	Memory Command Response
\$06	Rx	Execute Command

Figure 5.5: Serial Protocol — Command Packet Structure

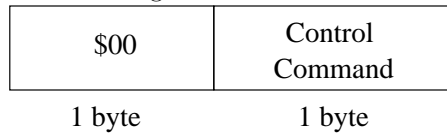


Table 5.2: Serial Protocol Control Commands

Command Number	Command Name	Description
\$00	REQ (R)	Request a packet to be sent from BERT to PC
\$01	ACK (TR)	The previous packet was received
\$02	NAK (TR)	The previous packet was bad, send it again.
\$03	ERR (T)	The command was erroneous and was ignored.
\$04	NPK (T)	No packets to send, try again later.

Figure 5.6: Serial Protocol — Bit Logging Array Element

Mode	Gate	Sync	POC0	Number of "1" Samples
1 bit	1 bit	1 bit	1 bit	4 bit

Figure 5.7: Serial Protocol — Event Logging Packet Contents

Time Stamp	Bit Count GATE ON	Error Count GATE ON	Bit Count GATE OFF	Error Count GATE OFF
2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

Event Logging Packets

The Event Logging packets are the main reason for the serial interface on the BERT. They will be sent either every bit or every N bits (where $N \geq 2$) or both.

The logging to be done every bit is placed into a large memory array, and is not transmitted until the array is full (or specifically requested to do so). The format of the elements of this array is shown in Fig. 5.6. As the bits arrive sequentially, there is no need to time stamp the data in the array.

The logging done at the end of each sample window creates packets of the form shown in Figure 5.7. As the packets may arrive in an incorrect order, a two byte time-stamp is included in each packet. This allows the PC to sort these packets into chronological order (if required).

Read Memory Command

The PC can request to read a segment of the BERT's memory with use of the Read Memory command (see Figure 5.8). The READ MEMORY byte count is limited to the maximum packet size less the packet header (19 bytes - 3 bytes = 16 bytes). If more bytes are requested, the BERT will only return this maximum number of bytes, and the onus is on the PC to re-request the remaining bytes. This will decrease the load on the BERT which only has a fixed amount of processing power available.

Figure 5.8: Serial Protocol — Read Memory Packet Contents

Address to Read	Byte Count
2 bytes	1 byte

Figure 5.9: Serial Protocol — Write Memory Packet Contents

Address to Write	Byte Count	Data to Write
2 bytes	1 byte	(Byte Count) bytes

Figure 5.10: Serial Protocol — Memory Response Packet Contents

Address Read/Written	Byte Count	Data Read/Written
2 bytes	1 byte	(Byte Count) bytes

Write Memory Command

To change values of EEPROM parameters or RAM values, a Write Memory command was included (Figure 5.9). The software in the BERT will decide what type of memory the address corresponds to, and whether it is writable. An attempt to write to a non-writable portion of memory will result in an ERR return code.

Memory Command Response

Both the read and write memory commands reply with a memory response packet (as shown in Fig. 5.10).

Execute Command

The Execute command (Figure 5.11) will allow a command to be run from RAM. The values of the 68HC11 registers D, X, and Y to be used in the routine are specified, then the actual code to run. Internal to the BERT, an RTS (return from subroutine) opcode will be appended to the code provided, and it will be run via an indirect subroutine call.

5.3.3 Implementation

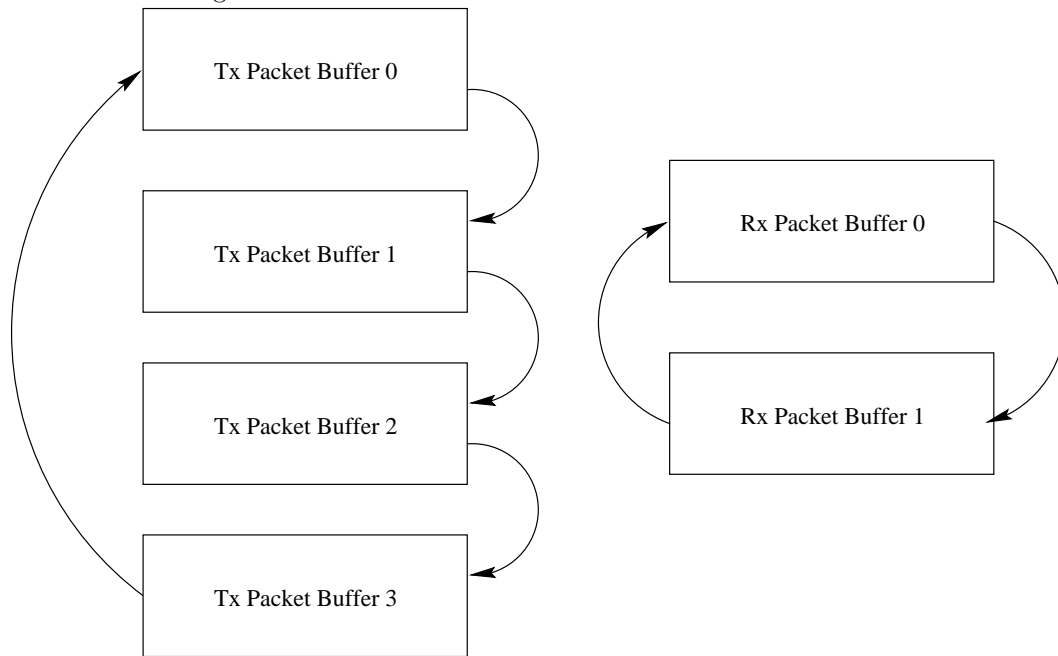
The software on the BERT to control the serial port is an interrupt-controlled background task.

As a starting point (for easier debugging), the BERT will simply queue the packets in a FIFO manner (say four Tx, two Rx packets buffered), and will

Figure 5.11: Serial Protocol — Execute Packet Contents

Accumulator D	Register X	Register Y	Code Bytes	Code
2 bytes	2 bytes	2 bytes	1 byte	(Code Bytes) bytes

Figure 5.12: Serial Protocol — Buffer Structure



only transmit one in response to a request from the PC. Eventually, all Event Logging and Sample Window packets will be automatically transmitted without any request to do so.

The buffers will be accessed in a cyclical manner. Thus, the Tx buffers would be used as 0,1,2,3,0,1... (see Figure 5.12).

5.4 Module Contents

Table 5.3: Source Files

Filename	Description
Assembly Source Files	
eeeprom.asm	Source code for EEPROM defaults
inputs.asm	Source code for input parameters and menu system
integral.asm	Source code for end of sample window error rate integration
intr.asm	Interrupt vectors
main.asm	Source code for mainline and miscellaneous utilities
outputs.asm	Source code for LCD screen and formatting routines
protocol.asm	Source code for Serial Protocol (BER to PC)
sample.asm	Source code for sampling algorithm and timers
serial.asm	Source code for Serial port physical layer
sync.asm	Source code for syncing algorithm
Include Files	
assemble.inc	Global assembler macros
ee_all.inc	Includes ee_comm.inc, ee_fixed.inc ee_var.inc
ee_comm.inc	EEPROM allocation definitions common to FIXED and VAR sections
ee_fixed.inc	FIXED EEPROM allocations
ee_var.inc	VARIABLE EEPROM allocations
eeintr.inc	EEPROM interrupt vector jump table
eejsr.inc	Definition of the EEPROM jump table
eeeprom.inc	Header file for EEPROM module
hc11k4.inc	Definitions of 68HC11K4 ports and bitmasks
hd44780.inc	Definitions of LCD commands and bitmasks
inputs.inc	Header file for INPUTS module
integral.inc	Header file for INTEGRAL module
main.inc	Header file for MAIN module
outputs.inc	Header file for OUTPUT module
protocol.inc	Header file for PROTOCOL module
reset.inc	Definition of reset parameters
sample.inc	Header file for SAMPLE module
serial.inc	Header file for SERIAL module
sync.inc	Header file for SYNC module
timing.inc	Definition of timing parameters

Chapter 6

Hardware Design

The circuit board for the BERT has four layers (the internal layers are V_{CC} and GND , and external layers both carry signals). The dimensions of the printed circuit board (PCB) was determined by the choice of enclosure (5 in x 8 in x 8 in) which had to accommodate both an LCD module and an analog meter on its front face.

6.1 Expandability

As in the software design, one of the salient points in the design of the hardware was the need for future expandability. Several features of the circuit board result from this requirement.

Several otherwise unused microcontroller I/O pins are extended to through-holes suitable to soldering wires to allow new inputs to be used without the necessity of soldering the wires directly to the pins of the microcontroller.

Eight of these inputs are buffered with CMOS buffer gates (MC14050) before the signals reach the microcontroller on PORT C. This will allow for signals that are sensitive to loading or that have a maximum level of less than 5V (the lowest logic “1” acceptable is just above 2.5V as a result of using the CMOS buffer). However, these eight pins can only be used as inputs or the buffers will be put into a state of contention which can destroy the buffer ICs and/or the microcontroller.

An additional four pins (the most significant nibble of PORT F) are brought out to through-holes, but are left unbuffered. This allows them to be used for either inputs or outputs.

To facilitate easy expansion of the circuitry, it is important to have space on the circuit board to add more ICs or other components. In this regard, an area of through-holes on a standard 0.100” grid is provided so that it is possible to add additional through-hole components.

Chapter 7

Realizing the Hardware

Concurrent to developing the software for the BERT, the hardware had to be designed as well. The schematics for the BERT were drawn up using the schematic capture software PADS-Logic (by PADS Software, Inc.) The layouts were created by a contractor using PADS-Perform (the sibling package to PADS-Logic), and were sent to a board shop to have circuit boards made.

Many revisions were made to the schematics before the layouts were created. This was necessary to allow for errors in schematic entry and to allow the design to evolve over time as the initial design requirements were quite general.

Before the layouts were created, it was necessary to source all of the components so that the layout of each component would be known at the layout stage. This included acquiring an enclosure that could hold both the LCD module and an analog meter on its front face. The internal dimensions of the enclosure dictated the shape and size of the circuit board.

To create the layout, an approximate parts placement was made (as the layout was to be done out-of-house). A life-size drawing of the circuit board was generated, complete with dimensions of the circuit board and its mounting holes. On this drawing, small pieces of paper cut out to the dimensions of each component were arranged to show a preferred layout of the parts. This drawing, when completed, was sent to the contractor along with the schematic files to obtain a complete layout of the circuit board.

When the contractor had completed the layouts, plot files (for an HP DesignJet 600) were returned to SRL. These preliminary plots were carefully checked for errors. The errors were noted, and the contractor created a final layout incorporating the fixes. Once again, plot files were transmitted to SRL, and were checked carefully.

The completed layouts were then transmitted to the board shop to obtain circuit boards.

Chapter 8

Testing of the Design

To test the operation of the BERT, it is necessary to have several pieces of lab equipment available.

A multiple-trace oscilloscope is required to view the waveforms during the testing. A digital storage scope is preferable for the majority of the testing as it is often helpful to be able to freeze the traces to diagnose the problem. The majority of the testing of the BERT at SRL was done using an analog scope with four traces (Tektronix 2445). Some particularly elusive problems were debugged using a digital storage scope with two channels (Tektronix TDS 320). The signals monitored were: `SYNC`, `DELAYED SYNC`, `POCO` and `CURRENTLY SAMPLING` (which is pulsed high for each sample taken). The combination of these signals (two inputs and two outputs), along with the behaviour of the analog meter and the LCD screen, will show enough information to allow most problems to be easily diagnosed.

Also required are a synchronous serial data source (including the serial clock) and demodulated radio data. During most of the testing the serial clock (`SYNC`) and the serial data (`DATA IN`) were tied together and were driven by a the TTL `SYNC OUT` signal from a function generator (both a Wavetek model 148A and an Hewlett Packard 3314A were used at different times). This represents the most simplistic mode of use of the BERT – being driven with a periodic square wave signal.

The demodulated radio data (`POCO`) was provided by a SRL pager radio (900 MHz band) that was being directly driven from a signal generator (both a Hewlett Packard 8656B and an Anritsu MG3632A were used at varying times). This allowed the variation of the Bit Error Rate by simply decreasing the signal level. This setup is virtually identical to the desired final use (the intended use requires the pager radio to receive the signal over the air rather than directly over a cable). Using the target radio from the start ensured that no unexpected problems would be discovered when first attempted.

8.1 Testing Sync–Sample–Compare

The first algorithm that was tested was the core operation — the Sync–Sample–Compare algorithm. If these functions did not operate correctly, the rest of the BERT was a futile exercise.

As the Sync, Sample and Compare algorithms are all interrupt–driven, testing them was not a simple task. The EVM will allow breakpoints in interrupt handlers, but it will not allow a user to trace through the RTI (return from interrupt) opcode, thus it is not possible to accurately watch the progress of the algorithm. In addition to this, the algorithm depends upon the timing between edges of the SYNC signal. When a breakpoint is hit and the EVM’s monitor takes over, many bit edges will be missed causing the time calculations (bit width and sampling times) to be totally incorrect. All debugging of this algorithm required creative (but logical) thinking and a thorough examination of the source code.

Initially, the algorithm was tested with no phase delay and in free–run mode as this shows the most simplistic case for the algorithm. The problems at this stage of the testing were the timing of the samples not being even, the samples not being centred in the bit and the occasional missed SYNC edge (caused by clearing the interrupt incorrectly).

Once the problems at this stage were eliminated, the phase delay was varied. This brought on a whole new dimension in the problems encountered. When the sample time occurred at the same time as a SYNC edge, often either the sample or the SYNC edge would be missed. The interrupt handlers were modified at this point to automatically pass from one interrupt handler to another without leaving interrupt mode when necessary to ensure that no interrupts were missed. Theoretically, this should never happen as the microcontroller was configured to use level–triggered interrupts, however the problem may be an idiosyncrasy of the EVM system.

Similar problems were encountered when the DELAYED SYNC timer triggered at the same time a SYNC edge was received. To further complicate matters, if the SYNC waveform had a duty cycle even slightly off 50%, the double buffering scheme failed. An elaborate workaround (as described in section 5) was needed to maintain a constant behaviour over all possible phase delays.

8.2 Testing Gated Mode

After the Sync–Sample–Compare algorithm was considered stable, the next logical step was to test the operation of the BERT in gated mode. When in this mode, the BER reported is the proportion of bits deemed in error while an external gating signal is held low. To allow for a gating signal that is active while at a logic high, circuitry was provided to invert the gating signal.

As the gating signal is likely to be coming from the radio, a user–selectable delay on the gate–on edge was implemented. This accommodates for the radio’s “ON–time” delay (typically in the range of $10ms$) so that the user can be sure that the data being checked for errors is coming from a stable radio receiver.

There is no delay on the gate-off edge (the rising edge as the microcontroller sees it) as the radio OFF-time is negligible.

8.3 Testing Serial Protocol

Unfortunately, as the workterm came to an end, there was not enough time left for a comprehensive test of the Serial Protocol.

Appendix A

Glossary

BERT	Bit Error Rate Tester
EEPROM	Electrically Erasable PROM
EVM	Motorola 68HC11K4 Evaluation Module
POCSAG	Post Office Code Standardization Advisory Group (of British Telecom) - referring specifically to the radio paging protocol they developed
PROM	Programmable Read Only Memory
SRL	Silcom Research Limited

Appendix B

Equations

$$q = 1 - p \quad (\text{B.1})$$

Probability of One Codeword Containing x Bit Errors given $p = BER$

$$P_1(x = 0) = q^{32} \quad (\text{B.2})$$

$$P_1(x \leq 1) = P_1(x = 0) + 32pq^{31} \quad (\text{B.3})$$

$$= q^{32} + 32pq^{31} \quad (\text{B.4})$$

$$P_1(x \leq 2) = P_1(x \leq 1) + \frac{32(31)}{2}p^2q^{30} \quad (\text{B.5})$$

$$= q^{32} + 32pq^{31} + 496p^2q^{30} \quad (\text{B.6})$$

Probability of Two Codewords Each Containing x Bit Errors given $p = BER$

$$P_2(x = 0) = P_1(x = 0)^2 \quad (\text{B.7})$$

$$= q^{64} \quad (\text{B.8})$$

$$P_2(x \leq 1) = P_1(x \leq 1)^2 \quad (\text{B.9})$$

$$= (q^{32} + 32pq^{31})^2 \quad (\text{B.10})$$

$$= q^{64} + 64pq^{63} + 1024p^2q^{62} \quad (\text{B.11})$$

$$P_2(x \leq 2) = P_1(x \leq 2)^2 \quad (\text{B.12})$$

$$= (q^{32} + 32pq^{31} + 496p^2q^{30})^2 \quad (\text{B.13})$$

$$= q^{64} + 64pq^{63} + 2016p^2q^{62} + 31744p^3q^{61} + 246016p^4q^{60} \quad (\text{B.14})$$

Appendix C

Schematic

Figure C.1 is the schematic of the Bit Error Rate Tester. As it was originally intended to be plotted on D-size paper (about 34 in by 22 in), all of the text is quite small.

Figure C.1: Bit Error Rate Tester Schematic

Bibliography

- [Bri80] British Telecom. A standard code for radio paging. Technical report, Post Office Code Standardization Advisory Group (POCSAG), November 1980.
- [Int86] International Radio Consultative Committee. *Recommendations and Reports of the CCIR*, volume VIII-1 of *XVth Plenary Assembly*, Dubrovnik, 1986.
- [Rhe89] Man Young Rhee. *Error Correcting Coding Theory*. McGraw-Hill, New York, NY, USA, 1989.