27
11-6-79
344711
SAND79-1668
Unlimited Release

UC-15

MASTER

An Overview of the Software for the Telemation/Sandia Unattended Video Surveillance System

Paul D. Merillat

Sandia Laboratories

SF 2900 Q(7-73)

SAND79-1668
Unlimited Release
Printed October 1979

AN OVERVIEW OF THE SOFTWARE FOR THE TELEMATIOM/SANDIA
UNATTENDED VIDEO SURVEILLANCE SYSTEM

Paul D. Merillat
Project Engineering Division 1759
Sandia Laboratories
Albuquerque, New Mexico 87185

ABSTRACT

A microprocessor has been used to provide the major control functions in the Telemation/Sandia unattended video surveillance system. The software in the microprocessor provides control of the various hardware components and provides the capability of interactive communications with the operator. This document, in conjunction with the commented source listing, defines the philosophy and function of the software. It is assumed that the reader is familiar with the RCA 1802 COSMAC microprocessor and has a reasonable computer science background.

DISCLAIMER

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	3
SYSTEM OVERVIEW	7
COMPENSATING FOR THE RCA 1802	
COSMAC ARCHITECTURE	9
PROGRAMMING CONVENTIONS	13
LISTING CONVENTIONS	15
INPUT/OUTPUT DEFINITIONS	16
THE TASK MANAGER	24
CONTROL PANEL OPERATOR INTERACTION	28
MOTION DETECT	34
ALARMS AND RECORDING	38
SUMMARY	42

AN OVERVIEW OF THE SOFTWARE FOR THE TELEMETION SINDIA UNATTENDED VIDEO SURVEILLANCE SYSTEM

SYSTEM OVERVIEW

The device is intended for use in fixed site nuclear facilities of the International Atomic Energy Agency (IAEA). Two cameras are used to monitor critical material handling areas in a facility. Video tape recorders will record scenes on video tape based on parameters specified by an authorized operator. Scenes are recorded at regular intervals. The system is capable of detecting motion within a scene as well as tamper and other unusual conditions. These events may also be recorded on tape. The scenes being viewed by the cameras are expected to be indoors and significant scene changes will consist of slow moving and persistent events.

There are three major tasks which are provided by the control software. The first is operator setup. This involves interactive communication with an authorized operator during which time the system parameters are defined. The second task

is normal operation. This involves the observation of the scene. The system, while in this mode, is constantly checking for significant scene changes. Scenes are recorded at regular intervals as well as during significant scene changes or other unusual events. The third task is to provide a video tape review capability to an inspector. In this mode, an inspector may review the recorded video tape in order to determine if any suspicious act has occurred.

The system is designed to run continuously for several months even during brief periods without AC power.

The decision to develop this system as a microprocessor controlled device is based on two features. First, the microprocessor's potential power enables many otherwise untenable features to be included in the design. The second is that a microprocessor system may be extended in capability and function in a reasonably straightforward fashion. In order to maintain this extensibility, the software has been carefully designed in a modular, top down fashion. Wherever practicable, table driven algorithms were used to facilitate the ease of future enhancements. A tailor made modular real time operating system was incorporated to further to enforce this goal.

The detailed documentation of the software consists of machine readable comments in the source listing. This document augments the source listing. The first section covers program conventions and philosophy. The second section provides details of the workings of the more complex aspects of the system. These aspects are the Task Manager operating system, the interactive operator interface, motion detection and alarm and recording methods.

COMPENSATING FOR THE RCA 1802 COSMAC ARCHITECTURE

The bulk of the programming conventions were developed in an attempt to compensate for the design restrictions imposed by the RCA 1802 COSMAC microprocessor. The 1802 was originally selected because of its low power requirements. The 1802 was the only well supported CMOS microprocessor available at the time of hardware commitment; however, the 1802 has several severe drawbacks to its architecture.

The general approach for this compensation was to develop an extensive Macro library which contains a wide range of pseudo instructions to augment the limited and awkward instruction set of the 1802. An 1802 assembler program which uses only the basic 1802 instruction set is noticeably difficult to follow because of the little work each individual 1802 instruction performs. For example, suppose one wishes to load one of the general purpose registers with a two byte variable. To load the two byte variable, XYZ, into register R9 would require the following 1802 code:

```
LDI HI(XYZ)    GET ADDRESS OF XYZ IN REGISTER R3
PHI R3
LDI LO(XYZ)
PLO R3
LDA R3         GET CONTENTS OF XYZ AND PLACE IN R9
PHI R9
LDN R3
PLO R9
```

This simple function requires 8 instructions and 10 bytes of code. If the programmer were required to use this sequence of statements each time he needed to load a register with a value from memory, two undesirable things happen. The first is

that the programmer is required to specifically test the eight instruction sequences each time he uses it. The second is that the source code becomes unwieldy in that only a few simple data manipulations are written per page of code thus masking the intent of the module being written. The Macro defined to perform this load task is used as follows:

1. R9,XYZ GET THE CONTENTS OF XYZ IN REGISTER R9

The Macro processor will generate the eight instructions for the programmer. The listing is produced without Macro expansion. Now all that need be tested is the Macro definition itself and the intent of the load function. Since this now takes only one line of listing space, the program becomes much more readable. This will also improve the maintainability of the system.

As such Macros have been defined. These Macros fall into two categories. These are:

1. Data transfer instructions.

These are load, store and move operations. The 1802 does not have an indexing capability. This has been added. For example, if the programmer wished to load the accumulator with a byte addressed by so many bytes past a base address, the LOIX Macro would be used. This Macro creates the desired address by adding the displacement to the base and loading off the new address.

2. Stack manipulation instructions.

The 1802 does not have a true stack nor does it have stack handling instructions. These have been added using register R2 as the system stack pointer. PUSH and POP Macros provide this service.

3. Compare and data modification instructions.

The 1802 has no compare instructions. Compares are done using subtract with a subsequent check for a borrow condition. The CI Macro aids in correcting this omission. Certain basic add and subtract functions are also implemented using Macros.

4. Subroutine transfer instructions.

One of the most awkward features of the 1802 is its handling of subroutines. The design of the 1802 dictates that high speed subroutine linkage is effected using the SEP command. A subroutine address would be loaded into a register other than that acting as the current program counter register. The program counter would be redefined as that containing the subroutine address using the SEP instruction. The return is performed by redefining the program counter back to the original one. In large programs this becomes unacceptable due to the fact that each subroutine must know what the program counter of the caller is. This prevents them from being general. Further, N levels of nesting would require N separate program counter registers.

At the expense of instruction execution time and object code space, the Macros CALL and RETURN were defined. These use a specially written caller and returner subroutine. The register use is:

- RD - Address of return point for the current subroutine
- RP - Program counter for the current subroutine
- RC - Address of the special caller/returner subroutine.

Subroutine calls are effected by:

1. Pushing the current return point (RE) onto the system stack.
2. Loading RE with the address of the subroutine to be called.
3. Setting the program counter to RC (calling the special caller/returner subroutine).
4. This special routine interchanges the contents of RE with RF and sets the program counter to RF.

Returns are effected by:

1. Setting the program counter to RC.
2. The special subroutine again interchanges the contents of RE with RF and sets the program counter back to RF hence returning control to the calling program.
3. The calling program POPS its return point back into RE from the system stack.

The instruction path length for this is 55 instructions. This, however, provides true subroutine capabilities which is imperative from the standpoint of development and maintainability.

In order for the Macros to work most of them require a free work register. This has been designated as R3 in this system and is not available for use by the programmer other than as automatically generated from the Macros.

The 1802 has a series of instructions known as short branch instructions. These branches, when they occur, only modify the

low order eight bits of the program counter register and hence their behaviour is dependent on memory page boundaries. This means that any module using these instructions is not relocatable. For this reason short branch instructions are not used except in one specifically located module loaded in low ROM immediately after the 9 byte STARTUP routine.

The input/output architecture of the 1802 is the last major drawback affecting programming philosophy. There are only 7 input/output ports on the 1802. This is not enough to handle the various external devices. In order to communicate with all the devices, it was necessary to multiplex the ports. This technique makes the I/O obscure. All I/O is done via 12 I/O Macros defined for reasons of clarity. See the I/O definition section of this report for detailed descriptions of the interfaces with the external devices.

PROGRAMMING CONVENTIONS

Register Usage

A convention of register usage has been followed for ease of development and maintenance. It is as follows:

- PC - This is the initial program counter used during STARTUP. It has not been used for anything else but may be used as a scratch register if desired during program maintenance.
- PC - This register contains the address of the timer interrupt routine and may not be used except as the program counter in the timer interrupt routine.

- R2 - This is the address of the next free element in the system stack.
- R3 - This is the Macro scratch register and may not be used by the programmer except as it is automatically used by Macros.
- R4-RB - These are general purpose work registers used by the Normal Tasks. The contents of these registers are saved by the Task Manager when a Task relinquishes control.
- RC - This register contains the address of the special caller/returner subroutine and may not be used except as the program counter for that subroutine.
- RD - This contains the address of the currently selected Normal Task's TCB. It may not be used for any other purpose. (See Task Manager description.)
- FE - This register contains the return point of the current routine. It may not be modified except by the special caller/returner subroutine and by the CALL Macro.
- RF - This is the normal program counter register and is used for only that purpose.

Normal Tasks may freely use R0 and R4 through RB. They may use R2 when they use the system stack. Tasks must ensure that the system stack pointer is as it was on selection when the Task relinquishes control back to the Task Manager. The contents of R4 through RB are saved for Normal Tasks by the Task Manager.

Rapid Polling Tasks may use R0, R4 through FB and RD but unlike Normal Tasks, R4 through RB are not saved for it. R2 may be used when the stack is required.

Subroutines may use R0 and R4 through R8. It is the responsibility of the subroutines to restore the original contents of R4 through R8 on return to the calling routine. R2 may be used for stack manipulations but the stack pointer must be the same on return as it was at call time.

The X register pointer may be changed as needed. Programs with instructions which use the X register must ensure the pointer is set up as desired before use.

Interrupts are disabled by STAKUP and enabled at the end of the initialization routine. Once interrupts are enabled, they are never disabled except during interrupt processing. The timer interrupts are the only interrupts in the system.

LISTING CONVENTIONS

The definitive and detailed documentation of the system exists as part of the source listing. This detailed documentation is, therefore, machine readable and must be changed when the source code is changed. Each Task, subroutine, and supervisor routine contains at the start of its listing a function definition block. This block contains:

1. Name of routine.
2. General functional description.
3. List of inputs.
4. List of outputs.
5. List of subroutines called.
6. Wait conditions (for tasks).
7. Date of original inclusion in the system and
8. Author.

The control Task Definition Block for a normal task remains after the function definition block.

In the body of the code, every attempt has been made to give the comments a clue to the intent and period of the code. The comments are numbered and ordered. It is anticipated that these comments be changed with the code is changed.

The system was developed on a Texas Instruments 8002 (a microprocessor for development of code). The system imposed an early restriction which, in order to save space in memory, had to be coding on a single line. The LINTLIB statement in the primary file directs the compiler to the macroprocessor. The Macro library is provided to each module via the LINTLIB statement. The Macro file names are "MACRO1", "MACRO2", and "MACRO3".

A file called "LIBRARY" contains the system reports. These are comments which may be subject to change at a later date. They have been collected in one place for ease of maintenance.

Unfortunately, the Texas Instruments 8002 does not produce a cross-reference listing.

INPUT/OUTPUT DEFINITIONS

Figure 1 shows a block diagram of the input/output devices which can communicate with the control microprocessor. The RCA 1802 COSMAC system has a limited number of I/O ports which makes the use of multiplexing necessary.

Below is a specification of the I/O sequences required to communicate with the external devices:

11. The Screen Control Array. The Monitor Array consists of an 16x4 character cells. Each cell may contain an ASCII character, a motion point, letter or number, A/D value and a specification of a color or intensity of color. The array appears as 16 lines of 40 to the control microprocessor. Control information is placed in the array via a four character code length instruction one cell. The first three characters are the instruction code. The remainder of the instruction is a 16-bit data area.

```

OUT 7  = 0 Select motion point
        = 1 Select color
OUT 6  = 0 Indicate motion point at top of this cell
        = 1 Indicate motion function in this cell
Data 1-4 These represent the last four bits of an
        ASCII character which will appear in
        the monitor.

```

Note that this array may be read as well as written.

12. Motion mode select. There are two types of motion mode. VISUAL mode caused a visible white bar to appear at the top of any cell which has a motion point turned on in the screen control array. AD mode will cause the video level to appear in the motion detect A/D for those points.

```

OUT 5 (data = 01) select AD mode
      (data = 00) selects VISUAL mode

```

 

[illegible]

 

-

information from the control panel station and the other communicates with the VPK controller. This switch may be set by:

OUT 3 (data = XX)

where XX = 00 selects the control panel VART and

XX = 01 selects the VPK controller VART.

At this point, OUT 1 command writes to the VART

control register to "0" (OUT 1 writes serial data out). This command tells the VART status register and INP 1 command reads VART status.

Either VART is interfaced with the sequencer:

OUT 2 (data = 00)

OUT 2 (data = 01)

8. VPK multiplexor (with 16 lines) the VPK VARTs are both selected, it may communicate with any of the three VPK controllers. This switch is set by the sequencer:

OUT 7 (data = 11) (this selects the VPK VART)

OUT 4 (data = XX)

where XX = 01 selects VPK 1

XX = 02 selects VPK 2 and

XX = 04 selects SLAVE VTR.

Before this multiplexor may be switched, it is necessary to read the VPK VART status register to be certain that it is not busy transmitting serial data (bits 7 and 6 must both be logic 1).

[illegible]

10. The following information pertains to the first year of operations of a company that has elected the LIFO method of inventory valuation. Assume that the company's beginning inventory is zero and that the company's ending inventory is 10,000 units.

[illegible]

when the file command is used, the VIP member will be reported with a general "data type" which is determined as follows:

- FILE - means the file exists
- FILE* - means the file is a FILE type
- FILE# - means the file is a FILE# type
- FILE@ - means the file is a FILE@ type
- FILE% - means the file is a FILE% type
- FILE^ - means the file is a FILE^ type
- FILE~ - means the file is a FILE~ type
- FILE! - means the file is a FILE! type
- FILE& - means the file is a FILE& type
- FILE& - means the file is a FILE& type

When the file command is used, the VIP member will be reported with a general "data type" which is determined as follows:

- FILE - means the file exists
- FILE* - means the file is a FILE type
- FILE# - means the file is a FILE# type
- FILE@ - means the file is a FILE@ type
- FILE% - means the file is a FILE% type
- FILE^ - means the file is a FILE^ type
- FILE~ - means the file is a FILE~ type
- FILE! - means the file is a FILE! type
- FILE& - means the file is a FILE& type
- FILE& - means the file is a FILE& type

- FILE - means the file exists
- FILE* - means the file is a FILE type
- FILE# - means the file is a FILE# type
- FILE@ - means the file is a FILE@ type
- FILE% - means the file is a FILE% type
- FILE^ - means the file is a FILE^ type
- FILE~ - means the file is a FILE~ type
- FILE! - means the file is a FILE! type
- FILE& - means the file is a FILE& type
- FILE& - means the file is a FILE& type

When the file command is used, the VIP member will be reported with a general "data type" which is determined as follows:

- 10.1.1.1. The slave device is initialized with the following:

```
000 3 (data = 0)
000 4
```

The behavior of the eight I/O devices is:

```
PIF 0 (data = 0) (data = 0)
PIF 1 (data = 0) (data = 0)
PIF 2 (data = 0) (data = 0)
PIF 3 (data = 0) (data = 0)
PIF 4 (data = 0) (data = 0)
PIF 5 (data = 0) (data = 0)
PIF 6 (data = 0) (data = 0)
PIF 7 (data = 0) (data = 0)
```

- 10.1.1.2. The slave device is initialized with the following:

```
000 3
```

The slave device is initialized with the following:

- 10.1.1.3. The slave device is initialized with the following:

```
000 7 (data = 0)
000 4 (data = 0)
```

THE TASK MANAGER

The Termination/Sandia unattended video surveillance system is a microprocessor based real time control application which will attend to several different external asynchronous events. If development of such a system is to proceed smoothly and the end product is to be maintenance free, an operating system must be implemented which is capable of handling real time asynchronous control functions. Such a system is a Task Manager.

A Task Manager is a program which oversees the orderly execution of independent program functions. These independent modules are called Tasks. Each Task in the system will be responsible for a specific function. A Task informs the Task Manager when wait conditions for the Task is to be activated for work. The Task Manager simply watches for these specified conditions to occur and when one does, the Task Manager directs the Task.

The Task Manager in this system handles two types of tasks. The first are called Rapid Polling Tasks. These are Tasks which by their nature require detection as often as possible. One example is the alarm watcher Task. This Task continuously monitors the status of the alarm register. The other Tasks are called Normal Tasks and require detection based on the occurrence of specific conditions.

There are two types of conditions a Normal Task may be waiting for. The first is the expiration of a certain amount of time. A Task in this system may perform some function and then request that the Task Manager re-select it some specified number of seconds later. As an example, the VTR record Task will power on the cameras if they were off due to loss of AC power. This Task will wait five seconds for the video to

slave line refers to starting a record sequence. This is effected via a time wait. User may also wait for a specific bit in memory to change. In this system, Task may wait for bit 7 of a byte in memory to be turned on (i.e., become logic 1).

This is called a traffic wait and the bit it is waiting on is referred to as the traffic bit. For example, the motion system block will place a motion point in the position counter memory, and then wait on the traffic bit in the variable MREADY. Bit 7 of MREADY is set to 1 by a Rapid Polling Task when the motion system hardware detects that the A/D motion value is ready to be read. Normal Tasks may also request that they be

terminated when either a certain amount of time has expired or when a condition (traffic bit value $\overline{0}$), which may occur during the time-out, combination wait.

Each Normal Task communicates with the Task Manager through a Task Control Block (TCB). Each Normal Task has its own TCB. Below is the definition of a Task Control Block:

Name	Location	Size	Function
WAITCT	Byte 0	1 byte	Specifies the type of wait being performed by the Task Bit 7 = 1 = Traffic wait Bit 6 = 1 = Time wait
TASKID	Byte 1	1 byte	An identifying number (basically not used)
TIMEOUT	Bytes 2-3	2 bytes	If this is a time wait then this field contains number of seconds remaining before selection
ADDR	Bytes 4-5	2 bytes	If this is a traffic wait then this field contains the address of the byte containing the traffic bit.

AENTRY	Bytes 6-7 2 bytes	This is the address of the next instruction in the Task to which control is transferred when reselection takes place
RSAVE	Bytes 8-24 16 bytes	This array contains the saved contents of registers R4 through RB when the Task relinquished control.

The initialization routine, INIT, collects an initial TCB from each Task in the system and places them in the Task Control Block Table. The group in which these TCBs appear in the TCB Table determines the Task's priority. If more than one Normal Task qualifies for selection at the same time then the one whose TCB appears earlier in the TCB table will be selected first.

The initial TCB for each Task appears as the first item in the Task. The initial TCB specifies the initial selection criteria and is defined via the Macro TCB as follows:

```
TCBNAME TCB      EPNT,ATRP,R4,R5,P6,R7,R8,R9,RA,RB,TIME
```

where:

- TCBNAME is the label of the TCB,
- EPNT is the label of the first instruction in the Task,
- ATRP is the label of the initial traffic byte,
- R4 through RB specifies the initial work register contents and
- TIME is an optional time wait parameter (in seconds).

When the initial selection criteria is met, the Task will be selected for the first time. Subsequent selection will be based on new selection criteria set up by the Task when it relinquishes control.

Control is relinquished back to the Task Manager via the Macro WAIT as follows:

```
WAIT TRAF,ATRF or
WAIT TIME,Rx or
WAIT COMB,ATRF,Rx
```

where: TRAF, TIME or COMB specifies the type of wait
(Traffic, time or both)
ATRF is the label of the traffic byte and
Rx is the register containing the number of
seconds for a time wait.

The label field of the wait Macro is available for use if desired.

The Task Manager requires the ability to detect the arrival of each new second. This is effected by an interrupt routine connected to a one hertz signal. Each second, the interrupt timer will update the time of day variables as well as inform the Task Manager of a new second via the flag, NEWSEC.

The Task Manager is activated by the initialization routine and retains supervisory control until a system reset or system power up occurs. A Task Manager cycle consists of:

1. Call all Rapid Polling Tasks.
2. If NEWSEC is non-zero, then decrement each time wait field for every Task doing a time wait and then zero NEWSEC. Don't decrement past zero.

3. Get the address of the first TCB in the TCB table.
4. If there are no TCBs left then go to step 1.
5. If this TCB is not doing a traffic wait go to step 7.
6. If the specified traffic bit is on then go to step 9.
7. If this task is not doing a time wait, then go to step 9.
8. If the time field is not zero then go to step 11.
9. Select the Task.
10. Go to step 1.
11. Get the next TCB in the TCB table.
12. Go to step 4.

CONTROL PANEL OPERATOR INTERACTION

Approximately fifty percent of the code in the system is directly involved with the operator interaction through the control panel. It is felt that this area may be re-defined for later versions of this system. This redefinition would be due to the need for added features or a response to subjective changes defined by the end user. For this reason, the method for defining the operator interaction sequence has been implemented in an extremely general fashion to minimize the difficulty of such a change.

The approach used is similar to that used by the compiler generator systems, particularly the META compiler generator group. Briefly, a table is defined which specifies all legal semantic sequences. The table also includes subroutine addresses such that when a legal action sequence is encountered, the specified subroutine is activated to provide the appropriate semantic processing. Error trap points are also provided so that when an illegal action sequence is encountered reasonable recovery is possible.

The number of legal sequences is infinite. For example, when the operator is adjusting screen split, he may press the arrow buttons as many times as he feels is necessary. This means that the table defining the syntax of the legal button pressing sequence must be able to refer to itself. The routine which processes this sequence with respect to the table is recursive (i.e., may call itself). This technique is fairly complex. The advantage to using it is that when a change is required, the table is changed and the sequence processing routine is left alone. The semantics routines may be changed, added or deleted without affecting the sequence processing routine.

The sequence processing routine is a generalized parser and editor in file "CONPT". This processor is a Normal Task which waits for new buttons to be processed. The syntax and semantics table is defined in file "SSDEF". This table is examined by the sequence processing routine in conjunction with the received button codes. Legal sequences will produce calls to the semantics routines. These routines appear in files "SEMR1", "SEMR2", "SEMR3" and "SEMR4".

The table which defines the syntax and semantics is produced by a set of three Macros. These are DFN, DDEFN and SEMN. The definition of each logical part of the table is preceded by a "syntax picture" to clarify the intent of the table. If the table is changed, then the syntax pictures must be changed as well.

Consider, for example, the legal sequence for the entry of the date. The operator starts this sequence with the DATE button. Once this is done, the operator may press numbers (to change the date field in the work line) or arrows (to move the date fields about the monitor). He exits this function by

pressing ENTER (to copy the new date from the work line to the display field) or by pressing any new function button (TIME, SPLIT, MOTION, etc. to exit without the copy). The syntax picture for this is:

[FD] :

Names in braces, { }, represent primitives (buttons). Names in brackets, [], represent other constructs. [FD] is the name of this construct. [ARROW] is defined as any of the four arrow buttons, [DIG] is defined as any of the 10 digit buttons and [FCN] is defined as any of the function buttons. Asterisks indicate error trap points.

A syntactically legal sequence is any sequence which may be found in a path starting at the left of the syntax picture and leaving at the right. Note the ability to have any number (including zero) of arrows or digits. An example legal sequence is:

DATE 7 9 0 7 2 5 UP UP LEFT LEFT ENTER

An example of an illegal sequence would be:

DATE 7 5 4 UP PAUSE

When an illegal primitive is encountered, the generalized parser will back up to the first error trap and wait for the next button. This provides the error recovery feature. If no

error trap is encountered during the backup, the construct returns an error condition to the construct which referenced the construct in error. When a semantics routine is called during the processing of a sequence, an error may be generated by the semantics routine. An example would be the routine COPY, which is called when ENTER is pressed, which returns an error condition if the date entered was not valid. Error backup works the same for semantic errors as it does for syntactic errors.

The table representing the syntax picture is defined by a combination of the three Macros mentioned earlier. The first Macro is DFN. This sets up an "AND" sequence. It specifies a group of constructs which must all appear and in a specified order. In the above example, the [FD] sequence is the primitive DATE followed by any number of digits and arrows followed by ENTER or a new function.

An element in this Macro is in one of four forms:

1. P,button,C - one occurrence of the specified button
2. P,button,k - any number (including zero) of the specified button
3. constr,C - one occurrence of the construct labeled constr
4. constr,k - any number (including zero) of the construct labeled constr.

The syntactic definition of [FD] is then:

FD DFN P,BDATE,C,DIGAR,k,EFCN1,C

where: DIGAR and EFCN1 are additional constructs.

In the `update` definition, we first add the error term and the semantic value of the subformula. The semantic reference may be added after any one semantic constraint and is in the form:

`if (not (isa semref 'semref)) (error 'update)`

The error trap may appear after a constraint specification or a semantic definition and appear as:

`if (t)`

`do (update) (error 'update) (define) with (error) (error)`
and semantic reference `semref`.

If `semref` is `[BICAP]` or `[PCAP]`, `[BICAP]` or `[PCAP]`, then:

`if (not (isa semref 'semref)) (error 'update)`
`if (not (isa semref 'semref)) (error 'update)`

In this Macro, the `update` routine is called with an "ed" constraint. This Macro is called with and returns a number of constraints of the form `semref` and `semref` and the "ed" constraint is added to the form:

`if (not (isa semref 'semref)) (error 'update)`
`if (not (isa semref 'semref)) (error 'update)`
`if (not (isa semref 'semref)) (error 'update)`
`if (not (isa semref 'semref)) (error 'update)`

If `semref` is 0, this indicates there is no semantic routine at this point.

`[BICAP]` and `[PCAP]` are defined with this Macro:

```

BEGIN:  BEGIN, END, AND, OR, NOT, XOR,

```

```

where:  BEGIN and END are "constructs" and AND, OR,
        NOT, XOR are "operators" and AND, OR, NOT, XOR

```

```

END:    BEGIN, END, AND, OR, NOT, XOR,

```

```

where:  BEGIN is the BEGIN statement, END is
        the END statement, AND, OR, NOT, XOR are
        the operators.

```

The third Macro for this definition is that which defines the "operator" construct. It is "op" and takes 1 argument, the name of the operator. [AND], [OR], and [XOR] are defined with this Macro:

```

AND:    MACRO  AND, ANDAN, ANDIF, ANDIFC
OP:     MACRO  OP, OP, OP, OP, OP, OP, OP, OP, OP, OP, OP
OR:     MACRO  OR, OR, OR, OR, ORAN, ORIF, ORIFC, ORAND,
        ORIF, ORIFC, ORIFC

```

The parameters of these definitions are all gotten coded and may be found defined in the file "OPROPR".

The generalized parsing routine starts by trying to satisfy the construct labeled "APR1". It does this by trying to match the input buffer to a legal "AND" or "OR" construct. This routine is recursive, calling itself whenever a construct refers to another construct. It has its own stack for this purpose based on register R8. The Macros POP8 and PUSH8 effect POPs and PUSHes on this stack.

MOTION DETECT

The motion detection task, MOTIONDT, is responsible for examining the video signal and determining if a significant change has occurred. This task is initiated after the TV signal has been input and is initiated whenever Z_{in} power is failed, the ZIN reset sequence is in progress or when the operator is behind the front panel.

When a significant change has been detected, a message is output to the ZB bus. The input is then cleared and the processor motion point is restored to the background state. A change in input/output status is when the TV signal is received. This point is cleared and is set to the background state. An output from a A/D converter, the A/D reset is passed to the background state via the A/D loop line. A background task continuously watches this line and sets the motion point to MPEAK when this condition is detected. MPEAK is a message to the operator indicating the motion has occurred. The MPE reset task is in progress.

Motion detect is controlled by the following parameters. These are:

1. MPMIN	-	Motion point sensitivity
2. BGRMIN	-	Number of seconds between background updates
3. MRCRCH	-	Number of significant changes during event check to constitute point motion
4. MCPS	-	Number of level checks per second
5. BGNS	-	Number of seconds required for a background update
6. EVNS	-	Number of seconds required for an event check
7. BGSHT	-	log base 2 of BGNS*MCPS

- 4. ALENMOT Number of points in motion to contribute scene motion
- 5. MOTLEN Number of motion points selected.
- 6. MOTTAB Index of motion point selected and number of points in scene in current event cycle.

The variables ALENMOT and MOTLEN are set by the operator during initialization. ALENMOT is the number of scene motion points which must be changed from the selected scene and scene motion points before motion is the default in the scene selected, which value the scene returns. MOTLEN is a value of points within the scene selected. The scene appears to each motion point the scene selected in the scene. Hence, if the scene selected motion point in the scene and selected motion scene selected from a scene motion would be generated at any of the 16 points significantly changed from the scene and reference. MOTTAB is a work value.

The position of the variables are set by the initialization routine, INIT, and remain fixed. These variables are changed if it was desired to make motion selections were differently than originally designed. This would be necessary if the unattended video surveillance device were to be used in an environment different from that originally designed. Presently, the motion scene checked for a motion with significant motion scene slow and persistent.

MOTLEN is the number of times each user selected motion point is sampled each second. This was originally set to 4. (This value is MOTLEN 4.)

In order to establish that the scene has changed, it is necessary to develop a set of reference points. This is called a background update. An update is performed when 1) scene

motion was just detected, and the system just finished a VIB record, or 3) after MBGCHG records have passed with no condition 1 or 2 having occurred.

An update is performed by taking level readings from each point for a total of EYMS records. This gives a total of BGRMS from MBPS samples. These values are averaged to establish a background reference level for each user selected motion point. Note that the more levels involved in the background average, the more confident we are of the reference.

An event check is done continuously after the reference level has been computed. Level readings are taken for a total of EYMS records. This gives a total of EYMS times MBPS samples for each event check. Originally background was set for 4 seconds (16 samples) and event checks lasted for 2 seconds (16 samples). Each time a new level is read its difference from the reference level is computed. If this difference exceeds MBHLS, then this is considered a change and is counted in the work table. If at the end of the event check there have been MBPCHG or more changes for a particular motion point then that point is considered to be in motion. If ALRMLV or more points are in motion, the alarm is considered to be in motion.

Determining the values for MBHLS, EYMS, BGRMS, MBPS, and MBPCHG depend on the environment expected and the types of motion being looked for. Different values produce different false alarm rates and different sensitivities.

The goal for the values originally selected are that the false alarm rate be very low (less than once per 10,000 event checks) and that the alarm would be sensitive to slow, long lasting motion. MBPS was originally set to 4 which represents the fastest the hardware can collect groups of 16 points.

The prototype hardware operating in a room with average indoor lighting would produce level readings ranging from 8 to 120. If the intensity at a point was not varying the values will change slightly. Assuming they vary according to a Gaussian distribution, the standard deviation is about 10%. The signal values for XNOSH and XNLS were 8 and 11, respectively.

The expected false alarm rate may now be computed. The probability that the A/B will not rise above a point not varying in intensity when either of the detectors is more than XNLS is a function of the standard deviation, σ . In our case XNLS represents 1 standard deviation from the norm and yields a probability of failure including a change of 0.0443. Assume for a false alarm to occur at a single motion point, 8 or more such false changes must have occurred in the 20 sample event cycle. In general, the false alarm probability for a single motion point is:

$$1 - \sum_{j=0}^{XNOSH-1} \frac{(1-p)^{S-j} (p)^j}{(S-j)! j!} \frac{S!}{j!}$$

where: S is the number of samples in an event cycle
 p is the probability the A/B returns a value differing from the reference level by XNLS or more

for our values the false alarm probability is 1.3×10^{-6} .

Some changes which can be detected are those in which the requisite number of motion points have changed; 12 parts in (10-8), or about 10% and have remained changed for a total of 2 seconds out of the five second event cycle.

These parameters may be changed so that the false alarm rate is higher but the points need only be in motion for short periods or that a minor change be detected for a long event cycle or for a major change to be detected for a very short event cycle. The environment of the system would determine the choices.

ALARMS AND RECORDING

There are three types of alarm which are returned on the Video Tape Recorder (VTR):

1. Initialization alarm.
2. Alarm detected and
3. Routine alarm.

Immediately after the operator has input the operational parameters into the system, two initialization scenes are recorded on each VTR. The first contains the date, time, identification of address input, number of motion points, the motion point alarm level, the normal record interval, the alarm record interval, the system identification code and the position of the date, time and alarm lines. The second initialization scene displays the position of the user selected motion points.

At this point, the displayed alarm counter is set to zero and the system proceeds with normal operation. New scenes will be recorded based on:

1. Motion (or scene change) alarms
2. AC power failures
3. Tamper alarms and
4. VTR failures.

The frequency of new records is based on the normal record interval, NI, the alarm record interval, AI, and the mode (FIXED or RANDOM). These values are set by the operator via the control panel buttons. The alarm record interval is used to inhibit records, limiting records to no more than once every AI minutes. This is intended to limit tape usage. If no alarm conditions are being sensed, then scenes are recorded once every NI minutes in FIXED mode. Routine scenes are recorded on the average of once every NI minutes plus or minus 50% when the system is in RANDOM mode.

Routine scenes are those in which no alarm condition was sensed since the last record. Alarm scenes are those in which an alarm was sensed since the last record. Alarm scenes will cause the VTR to PAUSE when they are encountered during VTR review providing the review mode is AUTO ANALYSIS. For this purpose, initialization scenes are considered alarm scenes.

Alarm scenes (other than initialization scenes) will have a character "T", "M", or "B" in the alarm line. The alarm counter in the alarm line represents the total number of tamper, motion or power fail conditions detected since initialization. When a scene is considered an alarm scene due to a tamper, the type of tampers are displayed as a set of two character codes near the top of the scene. These codes are:

1. C2 - Camera 2 tamper
2. C1 - Camera 1 tamper
3. V2 - Video 2 not present
4. V1 - Video 1 not present
5. SD - Slave door open
6. CD - Console door open
7. KA - Camera Tamper Runaway

Any time the AC power is off during a record (or failed since the last record), the character "B" will appear in the alarm line.

In normal operation, routine and alarm scenes are recorded first on the user selected primary VTR (VTR 1 or VTR 2). The scene is then recorded on the SLAVE VTR. If the primary VTR fails, then the scene will be written to the backup VTR and then to the SLAVE VTR. If the primary VTR failed during a routine record, the subsequent record to the backup VTR and to the SLAVE VTR will be flagged as alarm scenes. If the backup VTR fails, then the scene is written to the SLAVE VTR as an alarm scene. Scenes being recorded immediately after a VTR failure will contain a message near the top of the scene describing the failure.

VTR failures are detected if a VTR controller does not respond to a status request, the returned status indicates VTR has malfunctioned or the VTR status indicates the scene was not recorded properly.

After a scene has been recorded the next scene record will be initiated

1. After $N1$ minutes if no alarms are detected,
2. After $A1$ minutes if an alarm is detected within the first $A1$ minutes after the last record
3. Immediately after an alarm is detected if the alarm is detected after $A1$ minutes but before $N1$ minutes.

If the system is in *RANDOM* mode, $N1$ minutes is a random variable linearly distributed between $N1 - N1/2$ and $N1 + N1/2$ minutes.

If the AC power fails, the cameras are turned off which inhibits motion detect. Scenes will be recorded as described above with the exception that scenes will not be recorded due to motion. The resumption of AC power will cause the system to revert to standard operation.

Whenever the slave door is opened, no further scenes are recorded on the slave until the system is re-initialized by the operator. If a VTR fails, no further record attempts are made until the operator re-initializes the system. Slave VTR failures are not recorded on the primary or backup VTRs.

SUMMARY

The Telemation/Sandia unattended video surveillance system was developed as a microprocessor controlled device to enhance its power and extensibility. The software is highly modular and is run under the supervision of a Task Manager. Rigid programming conventions were developed to compensate for architectural limitations of the COSMAC microprocessor and to aid in development and maintenance of the system. This report presents the commented source listing.