Design Proposal to Upgrade the L2 Trigger to more than 128 Bits

R. Hirosky, M. Kopal, J. Linnemann, R. Moore, A. Yurkewicz

Abstract

Outlined here are the technical details of the proposed method to implement more than 128 trigger bits for the DØ level 2 trigger. This upgrade is needed to cope both with the new triggering strategies devised to accommodate the reduced L1 accept rate and to accommodate the extra trigger options available when the new silicon tracking trigger comes online. The design outlined here will only allow for a logical ORing of bits and is designed to be as simple as possible to commission within the existing L2 system.

Introduction

The L2 trigger system currently has a limit of 128 bits. This limit was based on the number of L1 hardware bits that are passed to the L2 system. The current L2 software has a single script object associated with each L1 bit. These script objects analyse the incoming L2 data for that event, using specialized tool objects, and then determine, based on the number and type of the physics objects that they find in that data, whether or not the event should pass. The present design has each script object associated with a single L1 hardware bit and corresponding to a single L2 output bit.

The system was originally designed to process a L1 accept rate of 10kHz and reduce this to 1kHz: these rates being, respectively, the maximum tracking digitization rate and the maximum calorimeter digitization rate. However the maximum L1 accept rate is currently far lower than originally expected at around 2kHz. As a result a new triggering strategy has been developed which effectively uses the OR of several L1 bits to dramatically improve turn on curves at L1 enabling higher thresholds and thus lower rates without loss of efficiency. However because of the tree-like structure of the original trigger design this uses up far more trigger bits that was envisaged. In addition to this pressure the new L2 silicon tracking trigger (STT) will soon be coming online. This was an extension to the original L2 proposal and will also require additional trigger bits at L2 to make full use of its capabilities. Without increasing the number of bit the L2 system will not be able to take full advantage of the STT capabilities nor will it be able to fully participate in the OR triggers which could potentially limit the L1 rate further because of insufficient rejection at L2.

The output bits from L2 are of two types: hardware and software. The 128 hardware bits are used to validate the incoming L1 bits, in fact the hardware masks the returned L2 bits with the original L1 bits. If the result of the masking operation is non-zero this triggers a full detector readout for the L3 trigger. The hardware bits are not stored anywhere in the readout data and thus, in addition to a hardware response, the L2 system writes the L2 bit mask into the output from the global processor. This is the bit mask that is used to steer the L3 trigger. Since the number of hardware bits cannot be easily changed any upgrade to the system must cope with returning 128 bits to the hardware as well as a more detailed bitmask to the L3 trigger.

Matters are further complicated by the necessity of handling special monitoring events: “unbiased sample” (UBS) and “forced write” (FW), as well as events with error conditions set. These cases are handled by additional software bit masks which have special interpretations based on the presence of monitoring and/or error bits in the header. As such these schemes will be unaffected by the addition of extra bits since the size of these bit masks is immaterial to their operation.

Current Software Design

The current design is implemented in the `processEvent()` method of the `GlobalWorker` class contained in the `l2gblworker` package of the DØ code repository. The core part of this code loops over the 128 L1 trigger bits and calls the `run()` method for the associated script object if the bit is set. In addition, for all the bits which pass there is a call to the script's `tagObjects()` method in order to tag all the physics objects which the script used. For normal (not UBS or FW) events only the tagged objects are passed to the L3 trigger. The simple calling sequence for a script that passes is shown in figure 1, with the return value for the `run()` call giving the single script result. The call to `tagObjects()` is only made if `run()` returns `true`.

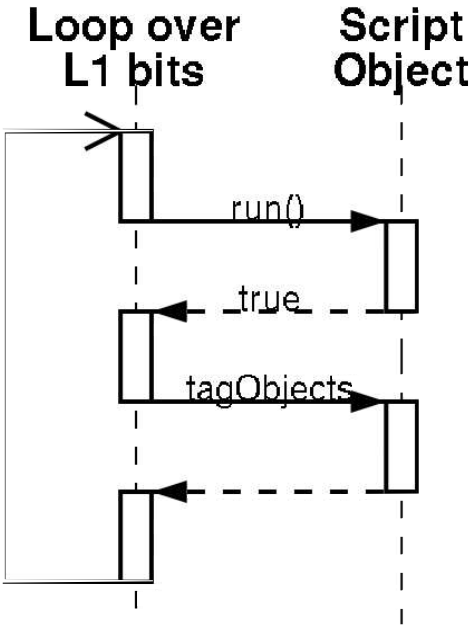

Figure 1:Calling diagram for the current script object. The `run()` method is only called if the corresponding L1 bit is set. The above cases shows a L2 bit passing.

In addition to running and tagging, the main loop over the L1 bits will also set the corresponding L2 bit in the output bitmask if the L2 script passes. For standard events this is the bit mask that is passed both to the hardware framework and the written into the output to the L3 trigger.

Upgrade Design Overview

The main issues to consider when upgrading the system to handle more than 128 bits are:

- The number of hardware response bits must remain fixed at 128 and so some mapping must be made from the true number of L2 bits to the original 128 L1 bits in order that a meaningful response can be given to the hardware.
- CPU time must be kept to a minimum. Expanding beyond 128 bits will require a more complex calling structure for the scripts and this should not add excessive amounts of CPU time, especially since the CPU time will now be divided between more scripts.
- The current system relies on the L1 bit mask to know which L2 scripts were run. Expanding L2 beyond 128 now means that care must be taken to ensure that we know exactly which L2 scripts ran for an event.
- Pre-scaling and its associated luminosity and monitoring implications need to be considered. In the current system each L1 bit can be independently prescaled or monitored removing the need for this in the L2 code.
- Monitoring of the increased number of bits: this does not rely on any fixed number of scripts or script mappings (except possibly at the display stage) and so will not need to be changed unless detailed information about individual L2 scripts is required.

- Scaling issues, such as output size limitations, may play a role if too many scripts are run. The current limitation for the example given is 255 output objects of any one type and, from the L3 header definition, a maximum of 256kB of data sent to L3 ($2^{16} \times 32$ bit words)

To address some of these issues this proposal will only discuss two scenarios: increasing the number of L2 bits to 256 or 1024. The current L2 global worker output format only has 8 bits for a script ID number thus 256 is a threshold value since increasing the number of scripts above this will result in a change to the script data format. In addition it is possible that there will not be enough resources at L2 to run significantly more than 256 scripts without either more CPU and/or major alterations to the output format to handle more than 255 objects of a given type. However if more than 256 bits will eventually be required it would be simpler to make a large increase to 1024 now rather than have two changes to the data format and code.

In either case scalability issues should be minimal when running on the Beta processors. These CPUs are several times faster than the original Alphas and there are already newer processors which are at least 2 times faster still. Also, testing with an early prototype of this scheme on the original Alpha CPUs showed a minimal additional overhead (on the order of approximately 1-2 microseconds for events with 2 bits set on average) so any CPU problems can be largely mitigated by reducing the number of L2 triggers configured as the effect on the Betas will likely only be significant for configurations with large numbers of scripts running for each event. The other resource which is affected by the number of bits is the output size and the effects of the two scenarios on this will be compared in detail later.

Prescaling

With the addition of branching at L2 prescaling also becomes an issue. Technically adding a pre-scale to L2 is trivial, and in fact a prescale tool used for commissioning tests already exists in the code base. This existing prescale tool should be sufficient to meet any prescale requirements at L2. Use of this tool will be identical to the L3 prescale where the prescale fraction is a parameter of the trigger list and does not vary until a new trigger list is implemented. Flexible prescaling is available at L1 and the L2 prescale is therefore neither intended nor required to account for luminosity variations.

Program Design

To allow the mapping of L1 to L2 bits to be fully flexible, two additional classes are required to replace the simple `Script` class in the global worker. These classes are `ScriptBase` which acts as an abstract base class, providing a common interface for the L1 bit loop to call, and `SuperScript` which acts as a container for normal scripts. The UML digram for this design is shown in figure 2.

This design would use the existing L2 parser to setup and configure the new `SuperScript` objects. In addition an array of 128 pointers is required to indicate which ScriptBase object should be run for each L1 trigger bit. This pointer array will be stored and configured in the global worker and would point either to `Script`s or `SuperScript`s depending on the mapping of L1 to L2 bits. The `SuperScript`s will implement the same interface to the main program as the normal scripts do but will, behind the scenes, call all the scripts which they are configured to point to. As a result the original program flow, shown in figure 1, is now replaced by the more complex flow shown in figure 3.

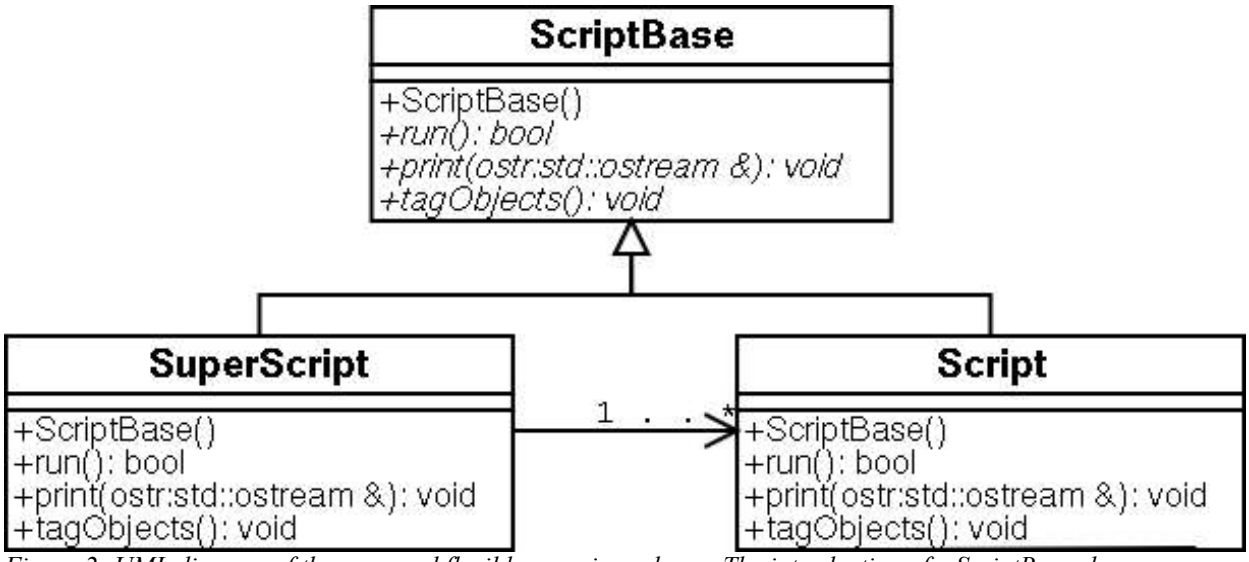

Figure 2: UML diagram of the proposed flexible mapping scheme. The introduction of a ScriptBase class means that the script interface can be abstracted so that the main program loop cannot tell whether it is calling a single script or a series of scripts.

This design has several key advantages over other possible implementations which were considered:

- No wasted scripts: the only constraints on the mapping is the number of SuperScript objects and the number of scripts which they can refer to. However these are software limits which are easy to fix and setting the number of SuperScript objects to 128 would remove all limitations.
- Scalable to more than 256 bits provided a change in data format to allow from script IDs greater than 256 is permissible.
- Minimal changes to existing code required and limited new code, namely:
  - Pointer array to `ScriptBase` objects in global worker
  - Write new `ScriptBase` and `SuperScript` classes
  - Make `Script` class inherit from `ScriptBase`
  - Add an extra configuration parameter to the `Script` class to allow for assigning an L1 trigger bit mapping
  - Alter the global header and script data output formats
- Possible to add logical ANDing as well as ORing (or any other type of combination) simply by writing a new type of `SuperScript` class.
- No immediate changes to code outside the L2 system should be required: these can be phased in over time

However the upgrade design has a few negative implications, in particular:

- Increased call overhead: `run()` and `tagObjects()` methods are now virtual function calls for all scripts plus the super script class must then call the same methods for its contained scripts (although this time using a standard, non-virtual function call). Early prototype testing on Alphas suggested that this is a minimal effect assuming ~2 L1 bits set per event on average.
- The trigger configuration is more complex:
  - Configure pointer array for scripts in the global worker which maps L2 scripts to L1 bits.
  - Configure the `SuperScript` objects.
  - Collect triggers with common L1 terms *and* L1 prescales into super script groups.

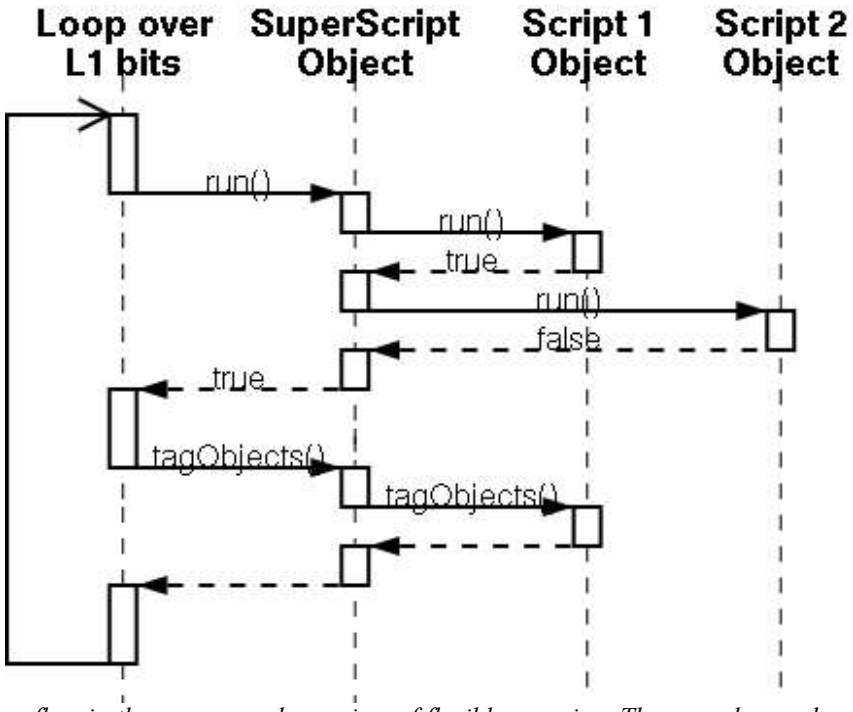

Figure 3: Program flow in the more complex regime of flexible mapping. The case shown above is for a call to a super script object which contains two normal scripts: one of which passes and the other fails. The super script is of the logical OR kind and so passes if any of its contained scripts pass.

Output Format

The changes to the output format to L3 which will be required to implement the upgrade are confined to the L2 Global header and script data formats. The current global header will remain unchanged but will be extended by adding an additional bitmask of the same size as the total number of L2 bits i.e. 256 bits (32 bytes) or 1024 bits (128 bytes) in size. There is also a slight change in interpretation of the bitmask called "`scriptBits`"[a]: currently this is understood to be the L2 scripts which were run and which passed. After the upgrade this now becomes the list of L1 bits which satisfy their associated L2 conditions and the new 256/1024 bit mask becomes the list of L2 scripts which were run and which passed, as shown in figure 4.

The format of the script data also needs to change. Firstly the associated L2 bit number is included so that consistency checks of the trigger result can be performed without resorting to the external configuration database. Secondly the size of the script ID field is increased to allow for more than 256 scripts. Finally the size of the number of objects field is increased by one bit using the spare bit from the L1 bit number. This will enable up to 511 objects to be associated with a given script compared to the current limit of 255. The description of the new script data format is shown in figure 5.

For the 256 bit case if the maximum number of objects remaining at 255 and the L1 bit mapping was dropped then the header could be reduced to 16 bits from 32 bits due to the smaller script ID field size

a To differentiate between the old 128 bit script mask and the new 256/1024 bit script mask the fields are called `scriptBits` and `fullScriptBits`. This confusion is unfortunately unavoidable since changing the name of the existing 128 bit `scriptBits` field would require all code referencing it to be changed as well, thus the name of the original field must remain unchanged despite the change in interpretation.

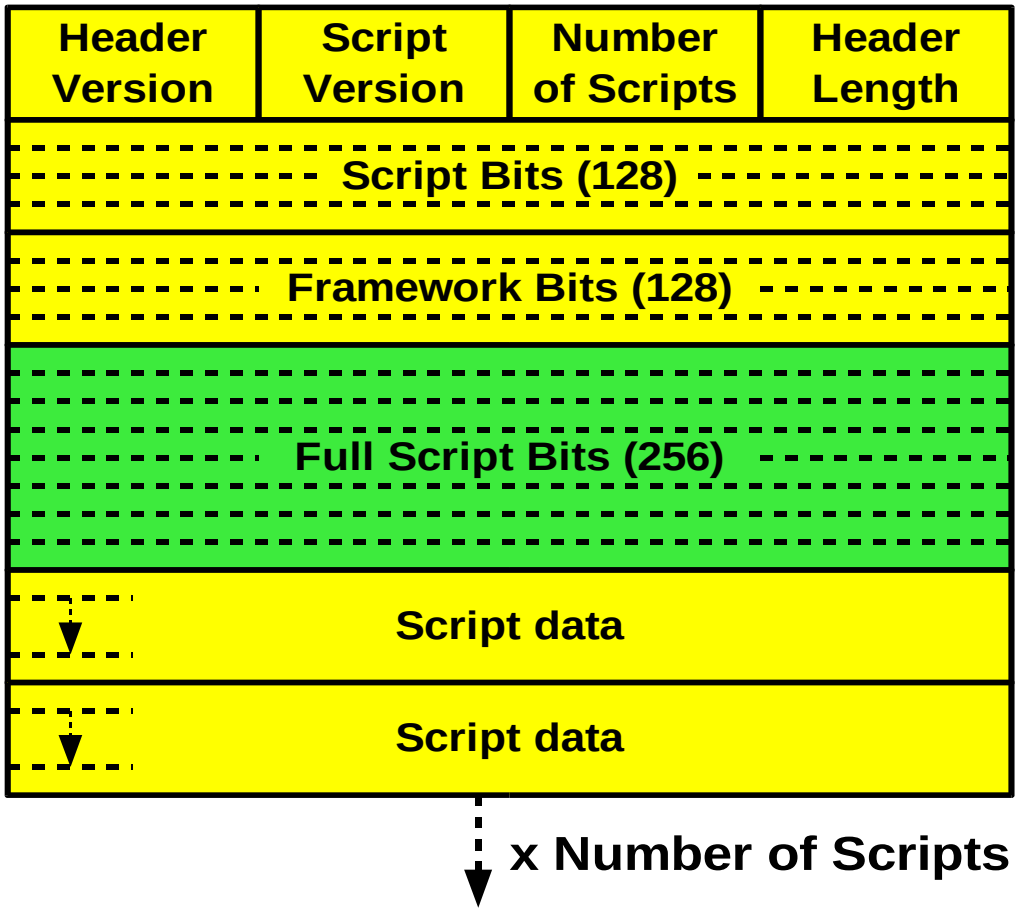

Figure 4: New format for the L2 Global header. The old "scriptBits" bit mask becomes the verified bits and a new bit mask of the same size as the total number of L2 scripts is added (shown here as the 256 bits option).The increase in header size is simply the size of the added bitmask which is 32 bytes for the 256 bit option and 128 bytes for the 1024 bit option, not including the increase due to the change in the script data format. The script data format is shown in figure 5.The dashed lines separate 32 bit words.

requirement. However this would remove the ability to check that the correct L2 scripts are firing for a given L1 bit. Thus the format shown in figure 5 will be needed in either the 256 or 1024 bit option and so one potentially significant advantage of the 256 bit option in terms of data size regarding the size of the script ID field, cannot be used[b].

The net effect of all these changes is to add on average 2 bytes per script passed and a fixed amount of either 32 or 128 bytes to the header.

b The header for the script data format must be an even number of bytes in order for the object IDs to be correctly byte aligned (and even then we require additional padding to get the correct 4 byte alignment in cases of an odd number of objects). Thus even though for the 256 bit case including the L1 bit mapping we could in theory reduce the header to 3 bytes we would have to pad it up to 4 bytes removing any potential space savings.

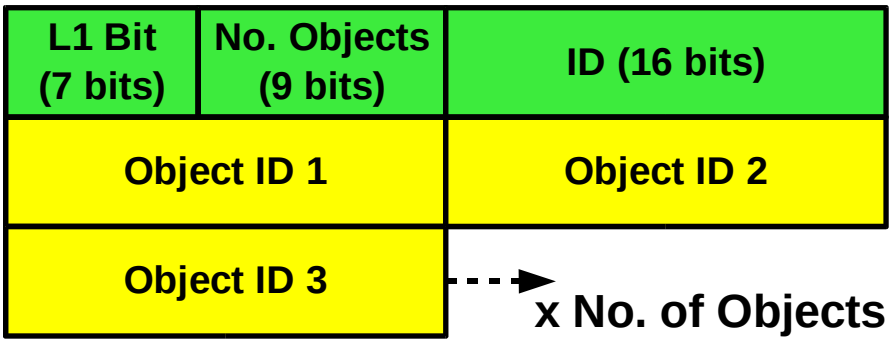

Figure 5: New format for the script data. The header increases from 2 to 4 bytes to enlarge the script ID field to 16 bits (from 8 bits), the number of objects field to 9 bits (from 8 bits) and to add a new 7 bit field to store the associated L1 bit number. Note that if there is an odd number of objects 16 bits of padding is added to the output.

Estimation of Increase in Event Size

Accurately estimating the increase in event size caused by the proposed upgrade is a very difficult task since the increase will be heavily dependent on the precise trigger configuration and beam conditions. Given this the best that can be achieved is an indication of the increase in event size for several possible scenarios. This is done in the table below for 7 different cases:

	Case A	Case B	Case C	Case D	Case E	Case F	Case G
Scripts/Event	5	10	40	128	128	256	1024
No. Bits	256	256	1024	256	1024	256	1024
Script data Increase (bytes)	10	20	80	256	256	512	2048
Relative Increase	**42**	**52**	**208**	**288**	**384**	**544**	**2,176**
Absolute Increase	**52**	**132**	**708**	**2,020**	**2,116**	**4,068**	**16,452**

The above table shows the expected relative and absolute increases in the event size in bytes for the events where the given number of scripts pass. The relative increase is the mean increment[c] compared to the current system's event size given the same number of scripts passing. The absolute increase is the increment relative to the current system's event size where 5 scripts pass[d] and there is an average of 5 objects per script. Two assumptions are made when calculating these values: that the number of physics objects found does not increase and that the number of objects per script does not increase. Both of these are reasonable assumptions since the number of physics objects found should depend mainly on what the detector sees rather than what the trigger is looking for and there is no reason to assume that the number of objects which each script requires to pass should be significantly higher with an increase in the number of trigger bits (in fact the reverse is likely to be true since the new ORing design philosophy uses simpler individual script requirements to remain generic).

Provided that these assumptions are met, in the worst case scenario (G) where all 1024 bits fire we would expect an increase of ~16kB in the event size. This is well below the 256kB maximum imposed by the data format and is still below the estimated mean event size imposed by the VME readout

c The script data is padded to be an even multiple of 4 bytes thus the actual increment in data size per script can be 0 or 4 bytes with the mean being exactly 2 bytes. This effect is what causes the difference in relative and absolute increases in the first column of the table.

bandwidth (estimated to be 20MB/s which gives ~20kB/event on average at a 1kHz event rate). However it is likely in this extreme case that the detector will be lit up like a Christmas tree and thus there will be significant numbers of physics objects which might cause the total event size to exceed this average bandwidth allowed for that event. Nevertheless if events such as these occur sufficiently regularly to be an issue then L2 bandwidth it is likely to be the least of our worries!

We expect that the actual scenario to be somewhere between case A and case B/C. The B/C cases were simply derived by taking the mean number of L2 scripts passing per event (5) currently observed in the data[d] and scaling that up by the same factor as the increase in the number of trigger bits i.e. factor 2 for the 256 bit upgrade and 8 for the 1024 bit upgrade. This assumes that the same fraction of scripts will pass. Clearly this is a conservative estimate since the scripts will not overlap precisely (otherwise we are unnecessarily duplicating effort) so the mean number of times each script passes should be less than in the current system. Hence a factor of 2/8 increase should be a conservative upper limit.

Finally it should be pointed out that should the fixed data size increase be an issue it is possible to drop the enlarged bit mask from the header without loosing information. The script numbers which pass are already stored in the script data and thus the bit mask can be reproduced from that. The reason that the enlarged bit mask is included is to simplify the conversion of programs which currently use the 128 bit trigger mask. It will be simpler to extend these to using a larger bit mask rather than recoding them to use the L2 script data but if an extra 32-128 bytes in the L2 header is considered an excessive price to pay for this simplification the decision could be revisited.

Configuration

The configuration scheme is designed to simplify the additional information that COOR needs to send to the L2 Global worker. The list associating L1 trigger bits to pointers to `ScriptBase` classes will be stored in a singleton class, `ScriptList`. Rather than require COOR to explicitly configure this list for every run, which would be a departure from our current policy of incremental configuration updates, an additional parameter, `L1BIT`, will be added to the `SuperScript` and `Script` configuration parsers. If this parameter is set then the `Script` or `SuperScript` will set the `ScriptBase` pointer in the `ScriptList` class corresponding to the given L1 bit, to point to itself. The parameter will be optional and if not set the class will not register itself in the `ScriptList`. This scheme enables COOR to update the L2 configuration without disturbing any existing configuration as is the case with the current code.

The configuration of SuperScript classes will trivially consist of a list of parameters SLOT1, SLOT2 etc. The values of these will be pointers to other Script objects. An example SuperScript configuration text is shown below:

```
L2GLOBAL WORKER SUPERSCRIPT2 {
     L1BIT=5,
     SLOT1=SCRIPT5,
     SLOT2=SCRIPT6,
     SLOT3=SCRIPT160
}
```

d Mean number of L2 bits set for events passing L3 in run 194272 taken on 18th June 2004.

This will configure a SuperScript to run when L1 bit 5 is set. The SuperScript will run three different L2 scripts (which would need to be configured elsewhere) and set bit 5 of the verified bit mask if any of the three scripts pass. In addition bits 5, 6 and 160 of the full L2 bitmask will be set if the individual scripts 5, 6 or 160 pass.

Commissioning

The commissioning of the upgrade must be designed to occur with minimal disruption to data taking, as such the upgrade is proposed to be undertaken in three separate phases.

1. **Simulation Testing (Optional)**
   Once the code is written and compiles the first stage will be to test it with the trigger simulator. This will involve editing the configuration file used by TrigSim to add the extra L1BIT lines previously described to map the L2 scripts to L1 trigger bits. This is an optional test which can be undertaken if scheduling delays the ability to go to phase 2.

2. **L2 Teststand**
   This phase will test the robustness and stability of the code. Initial tests will include configuring fixed pass or fail tools to generate known bit patterns for the full L2 script bit mask in order to ensure that the correct `Scripts` are called by their prospective `SuperScripts`. Once this is shown to work then by configuring the first 128 `SuperScripts` to run the first 128 `Scripts` the software can run in shadow mode processing real online data. This will give confidence that the code is both stable and performing correctly before running in production mode. The estimate is that, providing no serious problems are discovered, this will take a week of initial testing and running followed by a week of long term testing in shadow mode.

3. **Initial Production Running**
   At this stage we run the new L2 global code online but still limited to 128 trigger bits using the same trick that we used to run in shadow mode. Depending on the coupling of the online code to the L3 and offline production farms this could be as simple as updating the L2 executable to as complex as requiring both L3 and the offline production farms to upgrade to the new data format. This coupling is discussed in more detail in the following section, “Additional Considerations”.

4. **Full Scale Production Running**
   This is the stage where all restrictions are removed and COOR gets to configure the full capabilities of the new code. At this point more than 128 L2 bits can be configured by the system and used to trigger events. To reach this stage L3 will need to be changed to used the new, full script bit mask.

The estimated time line for writing and commissioning the code is, assuming that no significant problems arise:

- Implementing the code in L2: 1 week
- Configuring and running TrigSim tests: 2 weeks
- Running at the L2 teststand: 2 weeks
- Initial production running: 2+ weeks (until COOR and L3 ready for phase 4)

Additional Considerations

Coupling with External Systems

The upgrade to the L2 system will have implications on other systems outside L2: namely any system or program that reads the L2 data. The changes to the data format are designed to minimize any required changes. In particular the output data format has only been extended and all the original fields remain. Thus L3 and monitoring code which rely on the L2 bit mask at the most will simply need to be recompiled with the new object formats: field names will remain the same.

The coupling to external systems is even less if the official L2 code is being used to read the output data. This code uses the `L3Output` class which parses the `L3Header` object written in the data to parse the format. Thus, provided that the script data is not being used, no recompilation is required: this code will correctly decode the existing L2 128 bit masks and all other L2 data, with the exception of the L2 global script data which will be gibberish without a recompile.

However it is likely that both L3 and monitoring code will eventually want to take advantage of the expanded information available from the full 256/1024 L2 bit mask. Since the only change in this case is the size of the bitmask updating the code is expected to be a relatively trivial matter. Other systems which would have trouble accommodating a larger L2 bit mask can simply continue to use the condensed 128 bit mask. This is particularly relevant for the offline SAM header which has limited space available for the bit mask and so will likely remain using the original 128 bit mask.

The strongest coupling will be with the Thumbnail writer which has to read and interpret all the L2 data in order to store it in the Thumbnail. This writer is currently in the process of being rewritten and discussions are now underway to understand how to best integrate the changes required to interpret the new data format.

ORing Scripts

Recent DØ trigger strategies have involved the effective ORing of multiple L1 and L2 conditions. With the current tree-branch trigger design this has lead to the rapid multiplication in the number of trigger bits required at each level. The addition of branching at L2 now imposes a greater requirement on the number of L3 bits. For example an ORed trigger with 7 L1 conditions and 10 L3 conditions will require $7\times10=70$ L3 bits. If we now include 5 L2 conditions this becomes $7\times5\times10=350$ L3 bits! While a detailed discussion of potential solutions to this problem is beyond the scope of this document it is worth pointing out a few of the options and implications.

- The problem can be mitigated to some extent by having L3 simply continue using the original 128 bit L2 trigger mask which would provide an effective OR between the L2 conditions reducing the number of L3 bits back to 70. The disadvantage of this would be the inability to attach L3 scripts to individual L2 bits and it would also require that the trigger database attach L3 scripts directly to L1 bits and not L2 scripts as is currently the case.
- If the expanded number of L2 bits are to be used in a pure tree pattern (i.e. no L2 script can be assigned to multiple L1 bits, creating a loop) then additional configuration tools will certainly need to be developed to automate the vast duplication of scripts that will be required. Additionally user level code will be needed to simplify the ORing of the L3 bits. Creating a OR of 350 L3 bits

manually will be a very tedious task and prone to errors where bits are missed from the list.

- Allowing a loop structure (i.e. one L2 script can be attached to several L1 bits) would significantly reduce the number of L3 bits required but, at a minimum, will require a significant rethinking of the trigger configuration, particularly in regard to the trigger database. In addition the full implication of efficiency tracking would need to be very carefully considered.

In addition to the points laid out above the tree scheme also takes full advantage of a design feature of the tools used by the L2 scripts. These tools were specifically designed to be reused by multiple scripts in order to keep execution time down. Thus duplicate scripts using the same tools but hanging off separate L1 bits will not significantly increase the execution time. Each of the tools stores its results when it is first run on an event and then simply checks against those stored results when a new script asks.

Given these arguments this design will only implement a tree-like structure in the L2. The design does not address the management issues raised by the potential increase in the number of trigger bits but while extra management tools will be extremely useful and strongly recommended, in order to take full advantage of the L2 upgrade, they are not absolutely required.

Conclusions

The increase in the number of L2 trigger scripts to 256 or 1024 will allow for significant additional flexibility at L2 and in particular allow for the current ORed trigger design philosophy which was developed to cope with the reduced L1 accept rate and to take advantage of the new STT. The cost of this upgrade will be an increase of 32-128 bytes plus 2 bytes per passing script to the L2 global event output as well as an increase in L2 global's CPU budget which should be well below the additional capacity of the Beta processors for reasonable trigger configurations.

Commissioning the system should be possible without major upheaval of the online system. This is due to use of the L2 test stand to perform stability testing before running online as well as the flexible nature of the L2 data format which will allow for decoupled changes to programs reading the L2 data.

The fixed cost of the 1024 bit design over the 256 bit design is simply an additional 96 bytes in the output header format, which would be heavily outweighed by the size of the additional scripts which could be set for each event were all scripts fully configured. Thus, while the 1024 bit option might be capable of generating sufficient data that even the Betas might have I/O problems (though this is by no means certain and would require a large increase in the number of tagged physics objects), this can simply be fixed by reducing the number of scripts configured. Were the 256 bit option pursued then should 256 bits prove to be insufficient a later upgrade to 1024 bits would be an extremely painful process since all code using the L2 output would need to change at once i.e. we would be changing the size of the full L2 trigger mask and thus all code which used it would have to be updated simultaneously unlike the first implementation where code can migrate from the 128 bit mask to the full bit mask at leisure.

Thus, unless it can be demonstrated that we will never need to exceed 256 bits at L2 or the extra 96 bytes for each event that passes for the 1024 bit option would cause undue load on the Beta I/O, the 1024 bit option should be the one implemented.