CSR

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CENTER FOR SPACE RESEARCH
CAMBRIDGE, MASSACHUSETTS 02139

REVISION LOG

TITLE: Software Detailed Design
Continuous Clocking PRAM Builder Class

DOC. NO.
36-53235 Rev. B

Revision	Date (mm/dd/yy)	ECO No.	Page(s) Affected	Reason	Approval
A	4/18/96	36-596	all	Initial version. Incorporated comments from initial review. Uses **PramBlock** class (see 36-53234, ECO# 36-595).	RFG 4/30/96
B	6/17/96	36-623	953, 964	Removed initial CCD flush.	6/18/96

35.0 Continuous Clocking PRAM Builder Class (36-53235 B)

35.1 Purpose

The purpose of the Continuous Clocking PRAM Builder is to generate and load CCD-controller Program RAM instructions which clock the CCDs for the Continuous Clocking Science Mode.

35.2 Uses

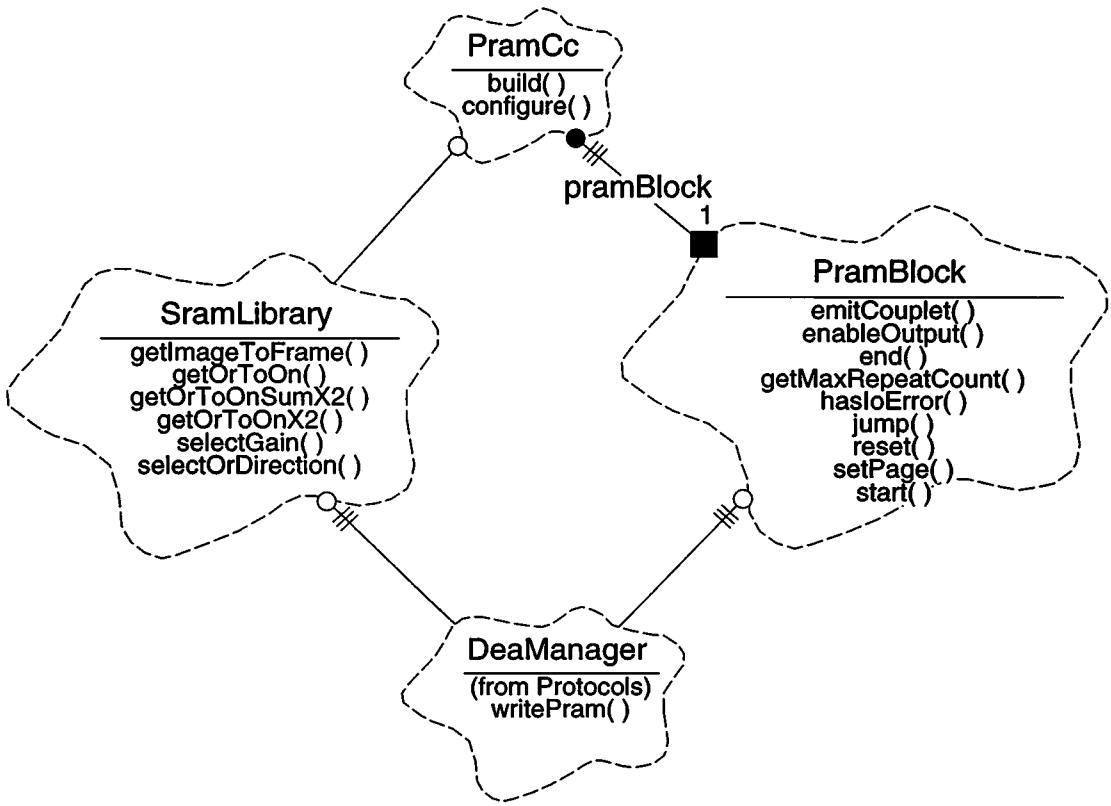
The following lists the uses of the Continuous Clocking PRAM Builder:

Use 1:: Build and load Program RAM instructions into a CCD-controller to perform Continuous Clocking mode CCD clocking.

35.3 Organization

The following illustrates the relationships used by the Continuous Clocking PRAM Builder class, **`PramCc`**. A detailed description of the **`PramBlock`** class is provided in Section 34.0.

FIGURE 161. Continuous Clocking PRAM Builder class relationships



PramCc - This class is responsible for generating and loading a series of sequencer Program RAM (PRAM) words needed to clock a CCD for Continuous Clocking mode. It provides a function which sets the desired clocking parameters (`configure`), and provides a function which generates and loads the clocking sequence into a particular DEA CCD-controller (`build`).

PramBlock - This class is responsible for emitting PRAM blocks to the DEA, each containing a header and one or more couplet entries. It is used by PRAM builders, such as **PramTe** (see Section 34.0) and **PramCc**, to load entries into PRAM. It provides functions which reset the instance to the start of PRAM and disable output to the DEA (`reset`), start and end a PRAM block (`start, end`), cause jump to a new PRAM page (`jump`), start writing to a new page within PRAM (`setPage`), emit a PRAM couplet (`emitCouplet`), obtain the maximum repeat cycles supported by a single couplet (`getMaxRepeatCount`), determine if an error was encountered while writing to the DEA (`hasIoError`), and enable output of PRAM words to the DEA (`enableOutput`). A complete description of this class is provided with the description of the Timed Exposure PRAM Builder classes in Section 34.0

DeaManager - This class is provided by the ***Protocols*** class category, and is responsible for managing access to the DEA interface. It provides a function which the builder uses to load words into a CCD-controller's Program RAM (`writePram`).

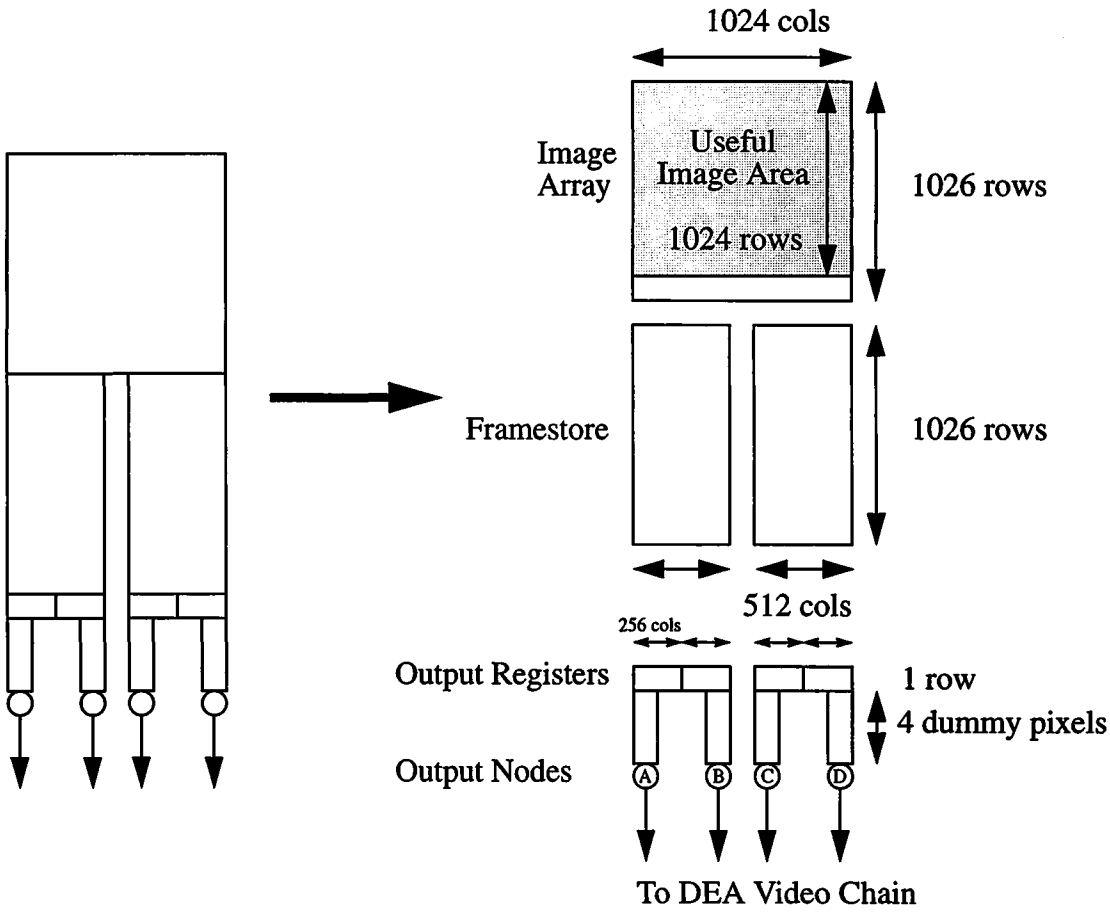
SramLibrary - This class is responsible for providing the PRAM builder with the Sequencer RAM (SRAM) locations of various primitives (NOTE: The function names abbreviate "output register" as "Or" and output node as "On"). It provides functions which select certain clocking options to use, such as the attenuated timing for the output register clocking, or the direction to clock the output registers (`selectGain`, `selectOrDirection`). It provides functions which supply the address of an SRAM block which clock and sample 1 pixel from the output register to the output node (`getOrToOn`), clock and sum two pixels from the output register to the output node (`getOrToOnX2`), clock and sum, without sampling, two pixels into the output register (`getOrToOnSumX2`). It also provides a function which returns the starting SRAM block and number of contiguous blocks used to clock one row from the image array to the framestore and from the framestore to the output registers (`getImageToFrame`).

35.4 PRAM Builder Design Issues

35.4.1 CCD Organization

Figure 162 illustrates a graphical representation of a single CCD. The figure on the left of the illustration presents a simplified picture of the main CCD components, and the figure on the right illustrates the pixel dimensions of each of the components.

FIGURE 162. Graphical CCD Representation

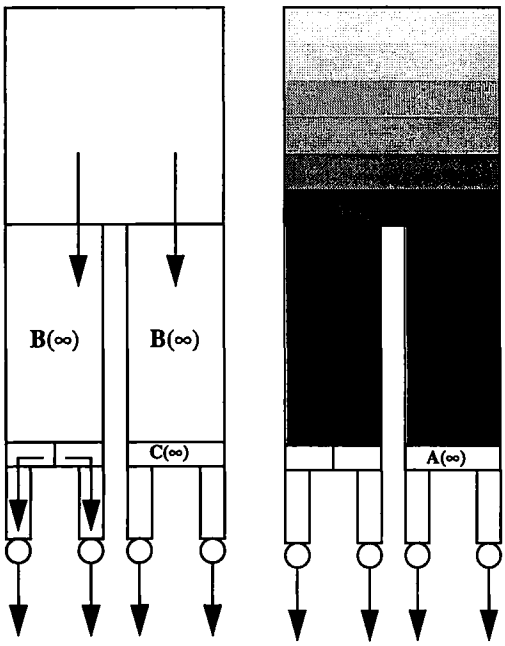


Each CCD consists of an Image Array, a Framestore, two output registers and four output nodes. Only 1024 of the 1026 Image Array rows are used for data acquisition.

35.4.2 Continuous Clocking Sequence

The following illustrates the sequence of operations used to clock out one summed CCD row in Continuous Clocking Mode (NOTE: For a description of the used SRAM primitives, see Section 35.4.3. The PRAM builder function associated with the action is posted at the bottom of each description):

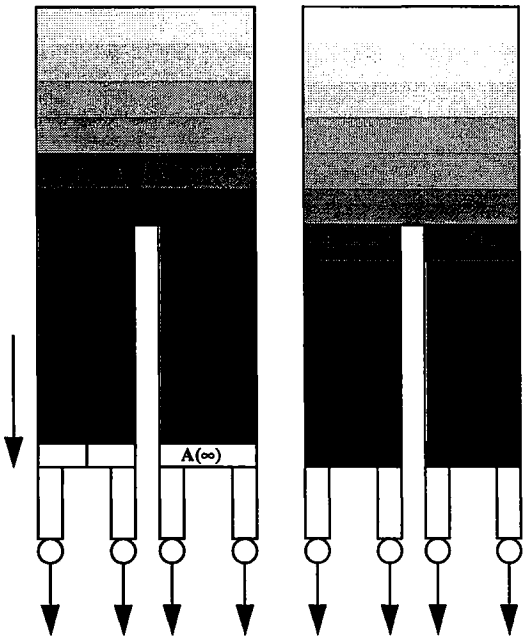
1. Discard all rows in image array and framestore



Repeatedly clock and discard 2052 CCD rows (1026 * 2), as described below by steps 2 through 5, to clear the contents of the image array and framestore, while establishing a constant integration time and noise profile of subsequent rows. The shading in the figure on the right is intended to represent the effective integration times of the rows within the CCD's image array. Although not shown in the figure, the exposure time of each row in the framestore to background radiation is a function of its position in the framestore. By the time a given row reaches the output register, the effective exposure to events and background is the same for rows preceding it.

(see `emitDataSet()`)

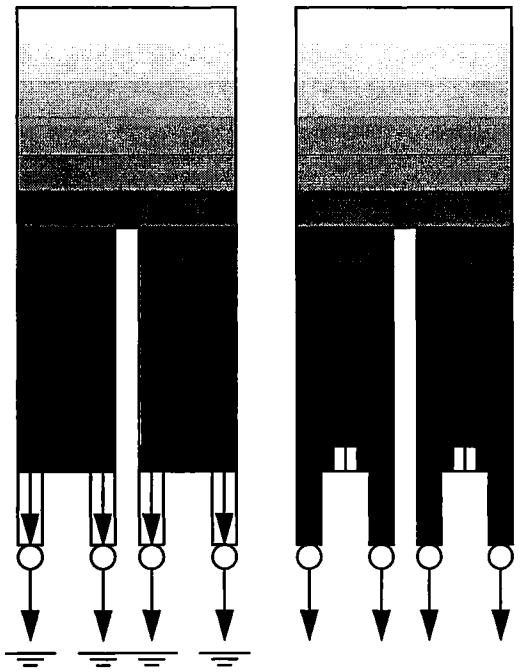
2. Clock and sum rows into output register



Concurrently, clock the desired number of rows from the image array into the framestore and from the framestore into the output registers. The rows clocked into the output register are summed.

(see `emitImageToFrame()`)

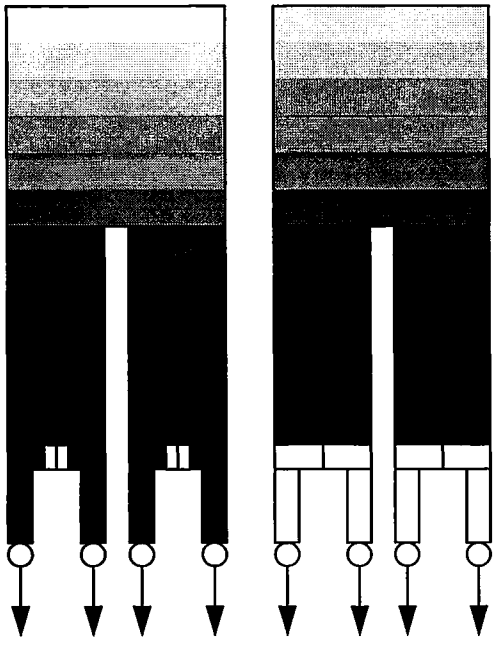
3. Clock and discard first four dummy columns



Clock four dummy column pixels from the output registers to the output nodes and discard the charge. On the last dummy pixel, emit a pixel code indicating the start of a data set.

(see `emitDiscardOr()`)

4. Clock row to DEA/DPA for processing

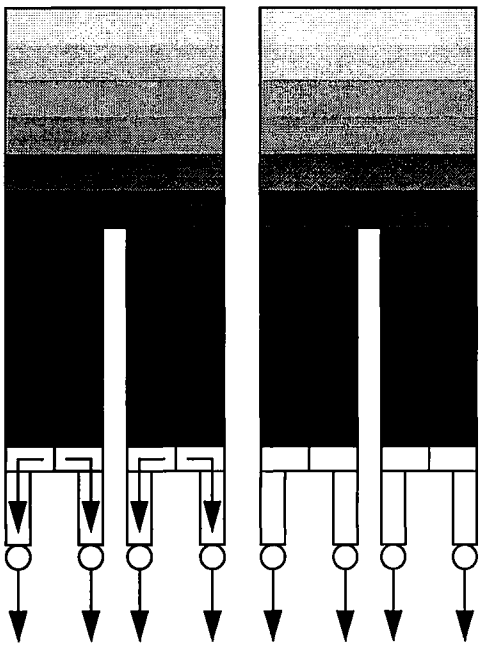


To DEA Video Chain

Clock 256 (512 if only using 2 output nodes) pixels from the output registers to the output nodes, sampling each and flagging the pixels to be processed. Meanwhile, the rows in the image array continue to integrate. NOTE: If performing on-chip summation, use an SRAM block which clocks and sums two pixels at a time to sum pairs of pixels at the output node. On the last pixel or pair of pixels, perform a final summation and sample the result.

(see `emitSummedPixel()`)

5. Overclock output registers

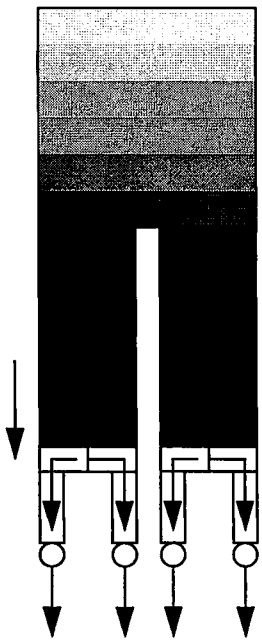


To DEA Video Chain

Clock the output registers a number of times, discarding the output, and then clock the output registers to produce the configured number of overclock pixels.

(see `emitDiscardOr()` and `emitSummedPixel()`)

6. Repeat 512 times to produce a complete data set



Repeat the steps starting from step 2 until an entire set of rows have been output. (NOTE: The number of rows output per data set is chosen to facilitate data processing and provide coarse timing markers. The number of rows does not affect the effective integration time of the rows, nor does it affect the continuous nature of the data). Continue producing data sets using steps 2 through 6 for the remainder of the science run.

(see `emitDataSet()`)

35.4.3 SRAM Primitives

SRAM consists of a collection of blocks, where each block performs a clocking operation. Some operations can be performed in a single SRAM block, whereas others, due to power constraints, require a series of SRAM blocks. Table 28 lists the SRAM operations used for Continuous Clocking Mode, where each SRAM block takes 1 pixel clock (~10μs) to execute.

TABLE 28. Continuous Clocking SRAM Operations

Operation	Number of SRAM Blocks	Description
Image to Frame	4	Clock one row from the image array to the framestore, and from the framestore to the output registers.
OR to ON discard (x2)	1	Clock two pixels from the output registers to their output nodes and discard the result.
OR to ON sum, x2	1	Clock and sum two pixels from the output registers into the output nodes.
OR to ON sample, standard	1	Clock one pixel from the output registers to their output nodes and sample the result (1 electron/ADU).
OR to ON sample, attenuated	1	Clock one pixel from the output registers to their output nodes and sample the result. Use signal timing to attenuate the gain (4 electrons/ADU).
OR to ON sample, x2, standard	1	Clock two pixels from the output registers to their output nodes, and sum and sample the pixels at the output nodes.
OR to ON sample, x2, attenuated	1	Clock two pixels from the output registers to their output nodes, and sum and sample the pixels at the output nodes.
Reverse OR to ON sum, x2	1	Clock and sum two pixels from the output registers away from their output nodes.[a]
Reverse OR to ON discard (x2 summing)	1	Clock two pixels from the output registers away from their output nodes and discard the result.
Reverse OR to ON sample, standard	1	Clock one pixel from the output registers away from their output nodes and sample the result.
Reverse OR to ON sample, attenuated	1	Clock one pixel from the output registers away from their output nodes and sample the result. Use signal timing to attenuate the gain (4 electrons/ADU).
Reverse OR to ON sample, x2, standard	1	Clock two pixels from the output registers away from their output nodes, and sum and sample the pixels at the output nodes.
Reverse OR to ON sample, x2, attenuated	1	Clock two pixels from the output registers away from their output nodes, and sum and sample the pixels at the output nodes.

a. The use of only two output nodes, such as in AC and BD mode, is selected via a discrete command to the DEA CCD controller to swap the output register phase clocks.

35.4.4 PRAM Headers and Couplets

This section describes the format of the PRAM words. PRAM consists of collections of word pairs known as 'couplets.' Each couplet contains the address of one SRAM block to invoke, a repeat counter indicating the number of times to repeat the block, and a pixel code, which instructs the Front End Processors (FEPs) how to process pixels being clocked while the SRAM block is being invoked. Each invocation of a couplet takes one pixel clock period (~10μs).

While PRAM is running, pixel clocks are always being delivered to the FEPs, regardless of whether pixel data is being clocked out of the CCDs or not. The pixel code portion of the couplets indicate how the FEPs interpret the data. Valid pixel data is flagged using a "Valid Pixel" code, and overclock pixels are flagged using an "Overclock" code. All other codes cause the clocked pixel data to be ignored. These include the "Vsync" code, which indicates that a new image has been started, the "Hsync" code, which indicates that a new image row is about arrive, and the "Ignore" code, which indicates that the clocked pixel should be completely ignored by the FEPs.

Each collection of couplets is preceded by a pair of 'header' words. The header consists of the number of couplets which follow the block, the number of times to repeat the collection of couplets, and an action to perform once the collection of couplets. These actions include: halt the sequencer, continue to the header immediately following the couplet collection, jump to the first PRAM location (restart), and jump to the indicated PRAM page.

The format of this header is as follows:

15	14	13	12	11	0
1	1	Option		PRAM Block Repeat Count	
1	0	Page Jump		Couplet Count	

PRAM Block Repeat Count.....This specifies the number of times to repeat the entire block minus 1

Option.....This specifies the next sequence option [0: Restart, 1:Continue, 2:Halt, 3:Page Jump]

Couplet Count.....This specifies the number of PRAM word pairs (couplets) following the block minus 1

Page Jump.....If Option is 3, this specifies the PRAM page to jump to

The following illustrates the format of the PRAM couplets within a block:

15	14	13	12	11	5	4	3	0
0	1	SRAM Page Address				0	PixCode	
0	0	0	0	Major Cycle Count				

PixCode.....This code is sent to the Front End Processor with each major cycle (pixel), [0:Ignore, 3:Valid Pixel, 4:End of Row (HSYNCH), 8:Start of Image (VSYNCH), 12:Overclock]

SRAM Page Address.....This specifies which block of 64 SRAM blocks should be sequenced during a major cycle.

Major Cycle Count.....This specifies how many times to repeat the selected SRAM block minus 1

PRAM Memory is organized around four contiguous pages, each containing 8192 words. The "jump" option of the PRAM Header causes PRAM to transfer control to the beginning of the specified page once the current PRAM block completes execution.

35.4.5 Clocking Algorithm

The following illustrates the clocking algorithm using pseudo-code, where the **bolded** text indicates operations which utilize the header repeat counter, *italicized* text indicates operations which can be accomplished using a single SRAM block and utilize the couplet repeat counter, and the underlined text indicates looping operations which are "unrolled" by the builder:

```
PRAM Page 0:
   /* ---- Clock and emit first row of data set ---- */
   Clock image array/framestore to sum "n" rows in output registers
   Discard 4 dummy pixels, and indicate start of data set (VSYNC)
   REPEAT for all summed pixels in output register
      Clock output register to sum "m" pixels. Sample result.
   ENDREPEAT
   Discard output register pixels prior to producing overclocks
   REPEAT for each overclock pixel sum
      Clock output register to sum "m" overclocks. Sample result.
   ENDREPEAT
   Indicate end of row (HSYNC)

   /* ---- Clock remaining rows of data set ---- */
   REPEAT for 511 rows
      Clock image array/framestore to sum "n" rows in output registers
      Discard dummy pixels
      REPEAT for all summed pixels in output register
         Clock output register to sum "m" pixels. Sample result.
      ENDREPEAT
      Discard output register pixels prior to producing overclocks
      REPEAT for each overclock pixel sum
         Clock output register to sum "m" overclocks. Sample result.
      ENDREPEAT
      Indicate end of row (HSYNC)
   ENDREPEAT
   Jump to PRAM Page 0
```

35.4.6 Parallel Transfers with Multiple CCDs

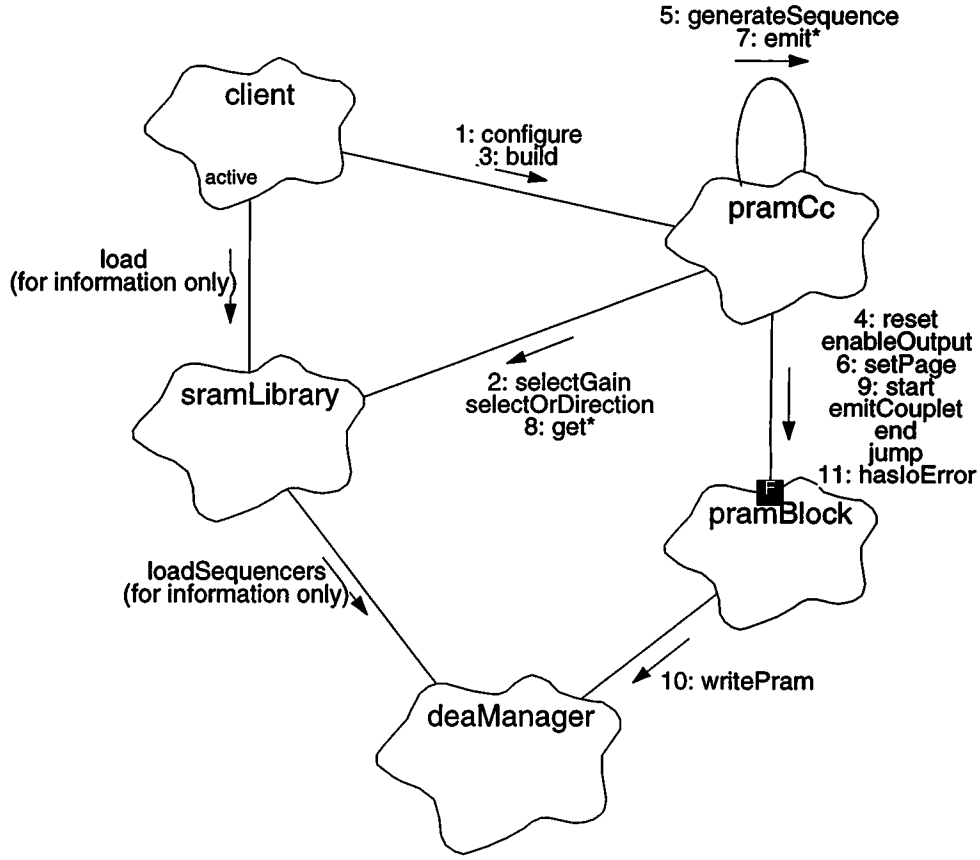
Assuming constraints on the maximum number of rows and columns that may be summed, phase-delays between image array/framestore row transfers on different CCDs, such as those used by Timed Exposure Mode (see Section 34.4.6), are not required.

35.5 Scenarios

35.5.1 Use 1: Build and load PRAM to perform Continuous CCD Clocking

Figure 163 illustrates the use of the **`PramCc`** class to generate a PRAM load for Continuous Clocking mode (NOTE: The `load` call to *`sramLibrary`*, and `loadSequencers` call to *`deaManager`* are for reference information only. For more detail, see Section 36.0). For a description of the use of the **`PramBlock`** class, see Section 34.5.1

FIGURE 163. Build PRAM Load

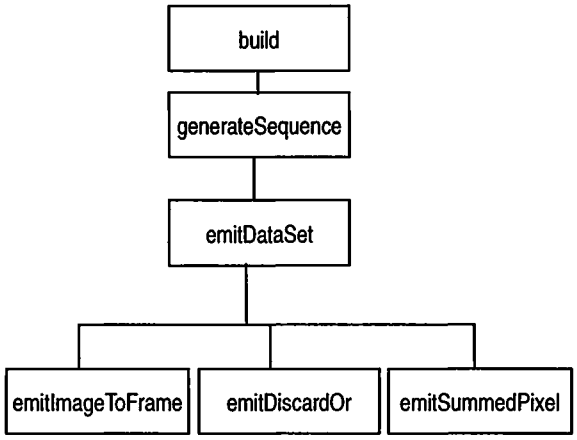


1. The *client* configures the builder, passing the clocking parameters, and pointer to the *sramLibrary* to *pramCc*.configure().
2. configure() initializes the *pramCc*'s instance variables and then selects which gain and output register clocking direction primitives to use, using *sramLibrary*.selectGain() and *sramLibrary*.selectOrDirection(), respectively.
3. The *client* then passes the CCD Id to *pramCc*.build() to generate the clocking sequence, and load the sequence into the DEA.
4. build() first invokes *pramBlock*.reset() reset *pramBlock*'s instance variables, and then calls pramBlock.*enableOutput*() to enable output to the DEA.
5. build() then calls generateSequence() to generate the clocking sequence and load it into the DEA.
6. generateSequence() uses *pramBlock*.setPage() to set the initial PRAM page to write to.
7. generateSequence() uses a variety of *pramCc*.emit*() functions to emit the various phases of the clocking sequence (see Figure 164 for a call structure chart).
8. The emit*() functions use the various *sramLibrary*.get*() functions to obtain the address and block count of the SRAM primitives used to perform certain functions.

9. The `emit*()` functions use *`pramBlock`*`.start()/emitCouplet()/end()/jump()` functions to generate the PRAM blocks (i.e. header followed by one or more couplets) needed to clock the CCD (see Section 34.5.1for the use of the **`PramBlock`** class).
10. The *`pramBlock`* functions use *`deaManager`*`.writePram()` to load the PRAM words into the DEA.
11. If an error is encountered during a write, *`pramBlock`* sets an internal I/O error flag and disables further writes, but allows the sequence generation to complete. Once *`pramCc`*`.generateSequence()` returns, the caller, *`pramCc`*`.build()`, then examines whether or not an I/O error was encountered using *`pramBlock`*`.hasIoError()`, and reports the condition in its return to the calling client.

Figure 164 illustrates a structure chart, indicating the functional hierarchy within the **`PramTe`** class:

FIGURE 164. PramCc Class Internal Structure Chart



35.6 Class PramCc

Documentation:

This class generates the Program RAM sequence needed to perform Continuous CCD Clocking.

Export Control: Public

Cardinality: n

Hierarchy:

Superclasses: **none**

Public Uses:

SramLibrary

Public Interface:

Operations:
`PramCc()`
`build()`
`configure()`

Protected Interface:

Operations:
`emitDataSet()`
`emitDiscardOr()`
`emitImageToFrame()`
`emitSummedPixel()`
`generateSequence()`

Private Interface:

Has-A Relationships:

unsigned *rowSum*: Number of rows to sum on-chip.This is copied by `configure()`.

unsigned *colSum*: Number of columns to sum on-chip. This is copied by `configure()`.

QuadMode *qMode*: Quadrant Configuration: Full, Diagnostic, AC, or BD. This is copied by `configure()`.

unsigned *overclockPairs*: Number of pairs of overclock pixels to output. This is copied by `configure()`.

Boolean *gain4*: Determines whether or not to use high-gain SRAM timing. If BoolFalse, use 1:1 SRAM, if BoolTrue, use 1:4 SRAM timing. This is copied by `configure()`.

const unsigned *arrayRows*: Number of non-summed rows in the CCD Image and Frame Store

const unsigned *nodeCols*: Number of non-summed columns per output node (assuming all 4 nodes in use)

const unsigned *dummyCols*: Number of dummy columns between the output register and the output node.

const unsigned *initialRows*: Number of rows to initially discard at beginning of run to achieve consistent integration times.

const unsigned *ocDummy*: This contains the number of dummy pixels to clock out of the output register prior to clocking out overclock pixels.

const unsigned *summedRows*: This is the total number of summed rows to clock out of the CCD per data set.

SramLibrary* *sramLibrary*: This points to the SRAM library to use when loading PRAM.

PramBlock *pramBlock*: This is used by the Continuous Clocking PRAM Builder to emit blocks of PRAM couplets.

unsigned *shiftCols*: This specifies the number of unsummed columns per output node (not including dummy columns).

unsigned *summedShiftCols*: This specifies the number of summed columns to shift out for each row.

Boolean *configured*: This flag indicates whether or not `configure()` has been called. If not, this field contains *BoolFalse*. If `configure()` has been called the field is *BoolTrue*.

Concurrency: Guarded

Persistence: Transient

35.6.1 PramCc()

Public member of: **PramCc**

Documentation:

This function is the constructor for the **PramCc** class. This function initializes the class instance variables, and sets *configured* to *BoolFalse*.

Concurrency: Guarded

35.6.2 build()

Public member of: **PramCc**

Return Class: **Boolean**

Arguments:

CcdId *ccdId*

Documentation:

This function builds a PRAM sequence and loads the sequence into the DEA CCD-controller specified by *ccdId*. The function returns *BoolTrue* if the sequence is successfully built and loaded into the DEA. It returns *BoolFalse* if an error was encountered while trying to write to the DEA.

Preconditions:

configure() must have been called (*configured* == *BoolTrue*).

Semantics.

Call *pramBlock*.reset() and then pass *ccdId* to *pramBlock*.enableOutput(). Build and load the sequence using generateSequence(), and check for I/O errors using *pramBlock*.hasIoError(). If an error was encountered, return *BoolFalse*, otherwise return *BoolTrue*.

Concurrency: Guarded

35.6.3 configure()

Public member of: **PramCc**

Return Class: **Boolean**

Arguments:

unsigned *rowsum*
unsigned *colsum*
QuadMode *quadrants*
unsigned *ocPairs*
Boolean *attenuate*
SramLibrary& *sramlib*

Documentation:

This function configures the Continuous Clocking PRAM builder. *rowsum* is the number of rows to sum, and *colsum* is the number of columns to sum per output pixel. *quadrants* indicates the output node configuration and clocking direction (Full, Diagnostic, AC, or BD). *ocPairs* indicates the number of pairs of overclock pixels to generate. *attenuate* selects the SRAM timing. If *BoolFalse*, use 1 electron/ADU timing, and if *BoolTrue*, use timing which produces 4 electrons/ADU. *sramlib* is a reference to the SRAM library to use when building PRAM.

Preconditions:

rowsum and *colsum* must be less than some TBD values (determined by TBD DEA power constraints).

Semantics.

Copy the parameters into the corresponding instance variables, configure the SRAM library's gain and output register direction, and compute derived clocking count information. Finally, set *configured* to *BoolTrue*.

Concurrency: Guarded

35.6.4 emitDataSet()

Protected member of: **PramCc**

Return Class: **Boolean**

Arguments:

unsigned *nrows*
Boolean *flush*

Documentation:

Emit PRAM instructions to produce 1 data set of continuously clocked CCD pixels. *nrows* specifies the number of rows to emit in the data set, and *flush* indicates whether or not to use the produced data set. If *flush* is *BoolFalse*, treat clocked data as valid pixels. If *flush* is *BoolTrue*, then ignore all clocked data. This is used to discard the initial contents of the image array and framestore at the start of the run.

Semantics.

Use two passes to produce the PRAM load for a data set. On the first pass, emit the PRAM instructions to sum and transfer the first row of the data set. The second pass emits a PRAM loop which transfers the remaining rows of the data set.

On each pass, call `emitImageToFrame()` to clock *rowSum* rows from the image array to the framestore, and from the framestore into the output registers. The rows are summed into the output registers. Then call `emitDiscardOr()` to discard the dummy pixels from the output register. On the first pass, flag the last dummy pixel with a VSYNC pixel code, to indicate the start of the data set. Once the dummy pixels are discarded, call `emitSummedPixel()` to sample *summedShiftCols* summed output pixels from output register. If overclocks are being used, call `emitDiscardOr()` to discard a number of pixels prior to clocking the overclocks, and then call `emitSummedPixel()` to sample (2**overclockPairs*) summed overclock pixel values. Delimit the end of each row by passing HSYNC to `emitDiscardOr()`.

Concurrency: Guarded

35.6.5 emitDiscardOr()

Protected member of: **PramCc**

Return Class: **Boolean**

Arguments:

unsigned *npix*
unsigned *pixcode*

Documentation:

Emit PRAM couplets which clock *npix* columns from the output register into the output node, each time, discharging the output node, effectively discarding the pixels. *pixcode* indicates the pixel code to use when discarding the pixels.

Semantics.

If *npix* is greater than or equal to 2, start by discarding pixels two at a time by fetching the SRAM block using *sramLibrary*->`getOrToOnX2()`, and emitting the couplet with a repeat count of *npix*/2, using *pramBlock*.`emitCouplet()`. Pass *pixcode* as the pixel code to emit while performing the clocking. Then adjust *npix* to contain the remainder (NOTE: Use of a remainder is a hook in case faster SRAM blocks become available).

If *npix* is not zero (in this case, it will be 1), get the SRAM block to discard 1 pixel using *sramLibrary*->`getOrToOn()`, and emit the couplet to repeat *npix* times, using *pramBlock*.`emitCouplet()` and passing *pixcode* as the pixel code to emit while performing the clocking.

Concurrency: Guarded

35.6.6 emitImageToFrame()

Protected member of: **PramCc**

Return Class: **Boolean**

Arguments:

unsigned *nrows*

Documentation:

This function clocks *nrows* from the image array to the framestore, and from the framestore into the output register. The rows clocked into the output register are summed with each other, and with charge already in the output registers.

Semantics.

Call *sramLibrary*->getImageToFrame() to obtain the starting index of the series of blocks which transfer one row from the image array to the framestore and from the framestore to the output register, and to obtain the number of adjacent blocks need for the transfer. If only 1 block is required, call *pramBlock*.emitCouplet(), passing the index of the block, and a repeat count of *nrows* to emit the PRAM instructions for the transfer.

If more than one block is needed and more than one row is being output, emit the series of blocks *nrows* times (NOTE: In this clocking mode the repeat block is used to count summed output rows, rather than rows being summed).

Concurrency: Guarded

35.6.7 emitSummedPixel()

Protected member of: **PramCc**

Return Class: **Boolean**

Arguments:

unsigned *npix*
PRAM_PIXCODE *pixcode*

Documentation:

Emit PRAM couplets which emit a series of *npix* summed pixels from the output register. Each output pixel is flagged with the *pixcode* pixel code, and consists of the sum of *colSum* output register pixels.

Semantics.

This function starts by obtaining the SRAM indices which sum and accumulate a pair of pixels, sum a pair of pixels and sample the accumulated value, and add a single pixel and sample the accumulated result, using *sramLibrary*->getOrToOrSumX2(), *sramLibrary*->getOrToOnX2() and *sramLibrary*->getOrToOn(), respectively.

It then enters a loop which iterates *npix* times. Each iteration sums *colSum* pixels and emits a couplet specifying an SRAM block using *pramBlock*.emitCouplet(). The SRAM blocks are chosen to minimize the number of cycles needed to perform the operation for each pixel. If *colSum* is greater than two, it emits a couplet which sums and accumulates pairs of pixels *colSum*/2 times. If *colSum* is even, it emits a final couplet which sums two more pixels and samples the result. However, if *colSum* is odd, it emits a final couplet which accumulates just one more pixel and samples the accumulated sum.

Concurrency: Guarded

35.6.8 generateSequence()

Protected member of: **PramCc**

Return Class: **Boolean**

Documentation:

This function runs through the PRAM generation process. The first portion of the function emits PRAM instructions which flush the initial contents of the image array, and the second portion calls `emitDataSet()` to emit PRAM instructions which produce 1 data set worth of continuous clocking pixels.

Semantics.

Generate the main clocking sequence in PRAM Page 0. Call *`pramBlock`*`.setPage()` to start writing to PRAM Page 0. Then call `emitDataSet()` to produce the PRAM load which clocks a single data set of 512 summed rows. Then call *`pramBlock`*`.jump()` to loop back to the start of PRAM Page 0 to repeatedly produce data sets.

Concurrency: Guarded