

rithms use this redundancy to determine whether a blocked beam is permanently disabled due to a faulty LED 24.

[0068] If both IrDA receivers 42 associated with an activated LED 24 provide an output profile indicative of a blocked light beam path, then the LED is likely faulty based on the premise that only one of the two light beam paths between an activated LED and its two associated IrDA receivers should be blocked at a time. The distance between the two associated IrDA receivers 42 and the assumption that the touch event is far enough away from the activated LED so as not to block both light beam paths support the premise. Cases where the touch event is close enough to the activated LED 24 to block both light beam paths are noted by the firmware through the blockage of both light beam paths of adjacent LEDs. If the LED appears to be permanently disabled, it is possible to report the error but still continue near-normal operation with the remaining functional LEDs.

[0069] System Operation

[0070] Power-On Self-Test (POST)

[0071] Upon reset of the touchframe system, the touchframe firmware does some basic housekeeping by setting up the stack pointer to point to the end of the SRAM. It then clears the SRAM to make it easier to examine stack behavior in case of debugging.

[0072] Power-On Self Test then begins by forming a register “bucket brigade”, shifting a couple of known patterns from register to register and verifying that the registers are able to store and recall the patterns correctly. Next, a CRC test is performed on the FLASH memory. CRC is a special check-summing algorithm that is substantially better than additive checksums at detecting errors in a long byte sequence. Finally, an SRAM Nair test is performed to prove the ability of the SRAM fabric to store and recall data correctly.

[0073] Initialization

[0074] The next task for the touchframe firmware is to initialize internal data structures in preparation for start of operations. The largest data structure is an array of beam state variables. With reference to FIG. 10, each LED 24 forms two logical geometric light beams 54 because each LED is detected by two IrDA receivers 42. It is possible, and very likely, that one of the two receivers 42 may “see” the beam from a particular LED 24 while the other receiver is blocked. For example, beams from LEDs 3 and 4 are blocked to the view of IrDA 1 but visible to the view of IrDA 2. A logical LED beam 54 is defined by both the LED 24 and the IrDA receiver 42.

[0075] Next the touchframe firmware initializes the internal processor timer Y1 (FIG. 8) used to generate the fundamental timed interrupt that regulates touchframe system behavior. This timer is initialized to a period of 100us, which means that as many as 800 processor instructions may be executed during each timer “tick”.

[0076] The internal processor UART 48 is initialized to a rate of 19.2 Kbaud. Various I/O ports are initialized to a benign state, interrupts are unmasked and globally enabled, and assorted variables are initialized. Finally, the built-in processor watchdog is synchronized and enabled, and processor execution enters its main operational loop.

[0077] Main Operational Loop

[0078] The main execution path for the touchframe system is comprised of three major actions; 1) checking for and responding to commands from the GUI CPU 22, 2) checking for blocked beams in sets of pairs of overlapping triangles, and 3) coordinate converting and averaging the results of any blockage.

[0079] GUI Command Decoder

[0080] The touchframe system detects and responds to a pre-defined set of commands that the GUI CPU 22 issues asynchronously from time to time. For example, the GUI CPU 22 may send the status requests to the touchframe system at ten-second intervals. Upon receiving a status request, the touchframe system gathers any error information, e.g., beams blocked for more than an acceptable period of time, and formats that information into a message to send back to the GUI CPU 22.

[0081] Commands from the GUI CPU 22 to the PCBA 16 are single 8-bit characters. Responses from the PCBA 16 to the GUI CPU 22 are composed of multiple 8-bit characters. Each response begins with a response-type character, followed by one or more data characters, and ends with a termination character of 0xFF. The PCBA 16 originates two response types unsolicited by a GUI CPU 22 command. These responses are Touch and Release events.

[0082] Touch Event: 0xFE, 0xXX, 0xYY, 0xFF

[0083] Release Event: 0xFD, 0xXX, 0xYY, 0xFF

[0084] where 0xXX represents the 8-bit X coordinate of the touch and 0xYY represents the 8-bit Y coordinate.

[0085] All other responses occur only when replying to a command from the GUI CPU 22. For example, the GUI CPU 22 may request an error report or touchframe system firmware version information.

[0086] Error Report Command: 0x32—Response: 0xF8, 0xNN, 0xEE, . . . 0xEE, 0xFF

[0087] where 0xNN is the number of errors being reported, and 0xEE . . . 0xEE are the error codes (if any). If there are no errors whatsoever, the response is: 0xF8, 0x00, 0xFF.

[0088] Possible error codes are:

[0089] 0x01—FLASH checksum error detected.

[0090] 0x02—SRAM error detected.

[0091] 0x03—Processor error detected.

[0092] 0x04—Failed beam(s) detected.

[0093] 0x08—UART error detected.

[0094] 0x09—Serial overrun error detected.

[0095] 0x0A—Invalid Command received.]

[0096] Get Version Command: 0x34—Response: 0xF6, 0xNN, “04-076530-85-X-”, “0000”, 0xFF]

[0097] where 0xNN is the number of versions being reported (for backward compatibility—always 0x01), and X is a revision letter specifying the version of the Touchframe firmware. The