Enabling Recipe Automation for Non SECS/GEM Semiconductor Assembly Machines.

Heong Chee Meng and Amin Mohd Sani
ON Semiconductor
Lot 122, Senawang Industrial Estate,
70450 Seremban, Negeri Sembilan, Malaysia.
cm.heong@onsemi.com, amin.m@onsemi.com

Abstract

In today's semiconductor manufacturing, recipe is a very important portion of the machine system. With that in mind, there is a huge demand to enable recipe automation. Recipe automation here basically covers the download and upload of a process program. The download refers to downloading from the host to the machine, while the upload is referring to transfer of process program from the machine to the host. For a non-SECS/GEM machine (i.e., one that does not support communication to the host) this will be a challenge. The main objective of performing recipe automation is to ensure that the correct recipe is being used during the bonding process. With this, we can eliminate mis-processing that is due to loading the wrong recipe. The automation will not only covers the download and upload portion, but it also covers the selection of the recipe, which is automatically being done. This will eventually reduce human errors.

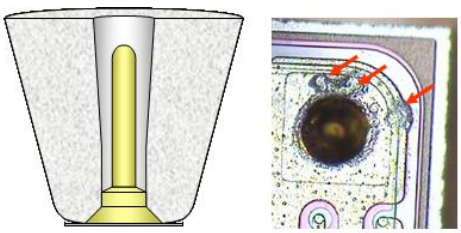
As we are all aware, non-SECS/GEM machine have no capability to communicate with the host at all, therefore we will need to come up with a method to somehow talk to the machine. This is actually the key point in making this project a success. If there is no way to communicate with the machine from the host side, then there is no way recipe automation can happen. Besides recipe automation, with this method also, we are able to retrieve crucial data that is needed. It is actually a 2 in 1 package whereby the machine is able to perform recipe automation and also auto data collection. Before the auto data collection, it is basically a manual process whereby the operator will have to look at the machine screen and manually record down the data that they want. These are crucial data or parameters that they need to monitor on a daily or per-shift basis. With the auto data collection, the data or parameters will be obtained via a web interface automatically. There is no longer a need to manually key in the parameters anymore on the paper. This can also help in terms of cost saving, as we no longer need to rely on paper. Besides that, with the auto data collection, the checking of limit is always enforced, if there is an out of control value, it will automatically raise an alarm.

This method is not generally applicable to all machines, but is very likely to succeed with many machines, especially those that use a general-purpose microprocessor (rather than a micro-controller) like the Intel 8088/80486, Motorola MC6809/MC68000, etc. A general-purpose micro-processor system has an exposed system bus. We propose to retro-fit a non-SECS/GEM system with a communications port (serial, parallel, etc) on the expose bus, insert new SECS/GEM communications code to utilize this new port and trigger this code using an unused interrupt. This assumes that there is enough IO or memory addressing space leftover, in addition to an unused interrupt. With most systems this is often the case. Indeed, most CPU boards contain empty sockets of communications ICs used by the original equipment developer. Where no empty port sockets are available, empty RAM or ROM sockets can be used. The cost of this total setup is basically around 1% of the total cost of buying a new machine. The saving is tremendous with this method.

With the recipe automation we are able to have better control on the process side, and indirectly it also increases productivity.

1. Manual Loading of Recipes

When there is no recipe automation, it will definitely result in the manual loading of recipes. Normally manually loading involves the user to select or press some buttons on the machine, and of course there will be a tendency for the user to select or press the wrong recipe. The manual loading of recipes can be extremely dangerous if a particular machine is running multiple devices. As in a normal case, 1 device is equivalent to 1 recipe. Imagine if the machine runs around 20 devices, which will be equivalent to 20 recipes. As we all know, human beings will tend to make mistake, maybe the 1st and 2nd time, there will be no mistake, but who can guarantee that by the 20th time of manual loading the recipe, it will be a correct one? This is the reason why recipe automation is so important. This is where RAM Emulator can help. However, if the machine only runs one device or one recipe only, then there is no need for this (no change of device).



Smear pad due to collision with capillary.
Root causes: wrong firing parameters & setup.

Figure 1: Examples of loading wrong recipes.

2. RAM Emulator

An EPROM Emulator is commonly used when designing or testing firmware for embedded controllers. Rather than repeatedly removing PROM/EEPROM or Flash device for programming, an Emulator is used instead. It has the advantage of a communications channel to an external development machine, which can download a program to the target machine in-situ. A RAM Emulator does the same for RAM, especially static RAM which is very often pin-compatible with EPROM or Flash. Instead of intercepting system ROM, it intercepts system RAM.

To retrofit SEMI Equipment Communications Standard / Generic Equipment Model (SECS/GEM) capability to old machines the first requirement is for some form of communications channel, in our case a parallel port. A more suitable channel might be the 3-wire SPI (Serial Peripheral Interface Bus). The new communications hardware require an entry-point to the target controller; a place with access to the recipe storage area. The latter can be floppy diskette, hard drive or most often in volatile memory, like RAM.

A generic solution requires this to work over many types of controllers with different CPU types. The entry point common to most controllers is RAM (Random Access Memory), specifically static RAM. The idea is to replace the original RAM IC/module with a RAM Emulator module, which in addition to having a communications channel, will also contain a dual-port RAM module of at least the same size. Ideally, the RAM location should also be that of the recipe storage area. While there is no guarantee that the recipe will be in RAM, there is nearly always a working copy in RAM.

This will enable an external computer to read or modify the contents of the recipe on-the-fly, without the target system CPU being aware of it.

The advantage of a communications-enabled RAM Emulator is we do not require the original equipment designer's involvement, and does not require software change in the target machine.

The ability to continuously monitor and modify the recipe on-the-fly enables us to intercept unauthorized recipe changes. Indeed, by adding more RAM Emulator modules where necessary this allows us to monitor much more than recipes: product counters, machine uptime, even sensor readings (sensor readings are very often first copied to RAM before use). In effect the RAM Emulator method allows us to retrofit most SECS/GEM functions to older machines.

The preferred architecture of a Static RAM Emulator is Dual-Port RAM and microcontroller. Many low-cost microcontrollers like those from Microchip Inc. are very low-profile, low-power, do not require support components and already contain serial port functionality. This is important as the parasitic RAM Emulator need to piggy-back on the target system RAM socket as a daughter-card and must not overload the power or system bus unduly. In addition it needs to be small enough to fit in the existing space, sometimes adjacent to other RAM Emulator modules.

In extreme cases where emulation of Dynamic RAM module is required, we will require in addition an FPGA/CPLD as a DRAM controller.

2.1 Locating Useful Data

We located the recipe memory area manually, by first installing a standard recipe into the target controller. Using the RAM Emulator, we uploaded all RAM contents for this our reference data set. Next we changed the recipe data one by one, uploading one complete set of RAM contents each time. Each time we applied a binary compare program to the data sets and the changes were recorded. We confirmed the data locations by modifying it manually, this time via the RAM Emulator and noting the change in the original program menu. Recipe data is normally in contiguous areas; and after very few tries the entire are can be located. There are potential problems with this method: the CPU nearly always continually changes the RAM stack areas, particularly if it runs a multitasking kernel or if it has interrupt handlers active. Some reverse-assembly of the original program code is necessary to confirm the locations of the CPU stack and the context-switch data areas. The other problem is there may be multiple copies of recipe data in RAM, and not every copy is current. By carefully changing the parameters in each area and observing machine behavior we can usually isolate the correct copy.

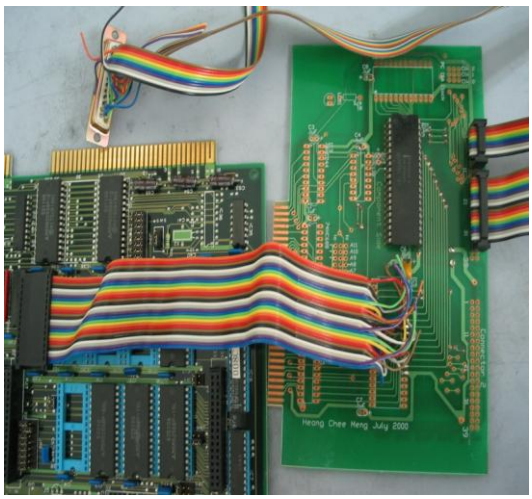


Figure 2: Left: target system (Equipment embedded controller). Right: RAM emulator prototype uses an Intel 8255 PPA (digital IO) IC for communications with a Linux PC's parallel port.

2.2 Linux Communications Controller

Because of portability, size and cost issues, the RAM Emulator works best with an external communications controller, with acts as an intermediary between the RAM Emulator and the SECS/GEM Server. The communications controller integrates serial communications to several RAM Emulator modules via its parallel port and handles communications with the CIM Server either in RS232C or TCP/IP. We chose to implement it as a 'headless' Linux WiFi router; that is a regular PC motherboard with only CPU,

DRAM WiFi and flash drive installed. The keyboard, mouse, hard disk, monitor and all daughter-cards were excluded.

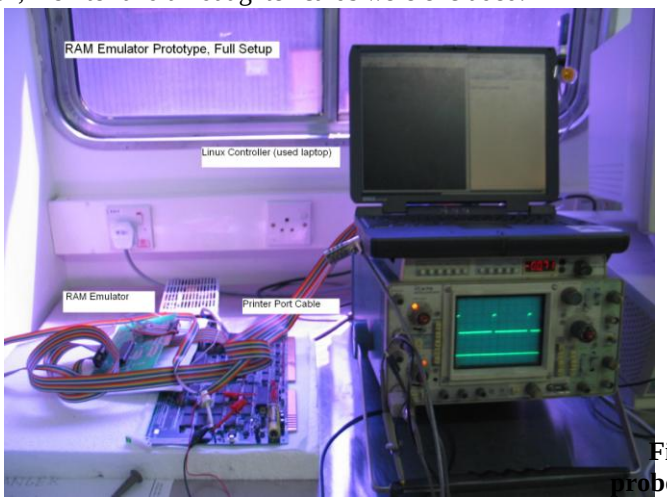


Figure 3: Prototype setup. Left to right: RAM Emulator, Equipment Controller, Laptop as Linux Controller. For full setup, insert controller into host.

Using PC parts enabled us to reduce hardware cost, and using Linux eliminated software purchases. A 'headless' configuration reduced the system footprint as space is often an issue with old machines. A standard Linux distribution, Slackware 11.0 was found sufficient. Indeed, Linux is often the most suitable operating system. For instance the Linux Samba application allowed seamless integration with Windows-based SECS/GEM Servers. Linux allowed us to quickly build bash scripts and programs to implement the SECS/GEM protocol in full, if necessary. Slackware also came with a full network routing suit, which allowed us to implement a backup adaptive routing back to the SECS/GEM Server in case the main Access Point should fail or be out of range. By using only bash scripts it is possible to reroute via neighboring Linux Communications Controllers. Ideally one of these should have a copper LAN connection to the SECS/GEM server, in case of complete failure of the main Access Point. The other critical component worth mentioning is the Linux WiFi device drivers for certain controllers can implement WiFi adapters as Access Points.

2.3 When a RAM Emulator Is Not Appropriate

In certain cases, the RAM Emulator may not be the simplest solution. Many embedded controllers already contain communications ICs and even associated firmware. In these cases it is merely sufficient to connect the Linux Communications Controller to these channels but present the same uniform interface to the SECS/GEM server.

We look on the 'RAM Emulator' as a concept: it can be applied to devices other than RAM, even if they are not in the memory map. Sometimes for space or PCB assembly reasons (e.g. large number of surface-mounted Static RAM ICs involving much manual rework in retrofitting, see Figure 4) it may be necessary to emulate an existing IO port IC, for example(see Figure 5).

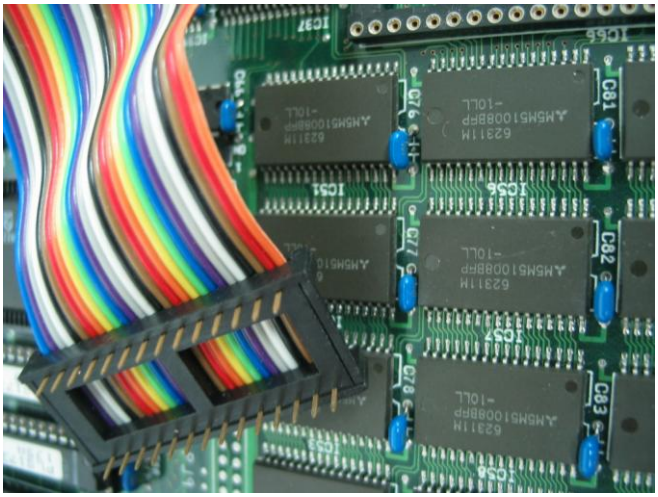


Figure 4: Mechanical incompatibility between DIP-28 probe and SMD RAM packages in the controller

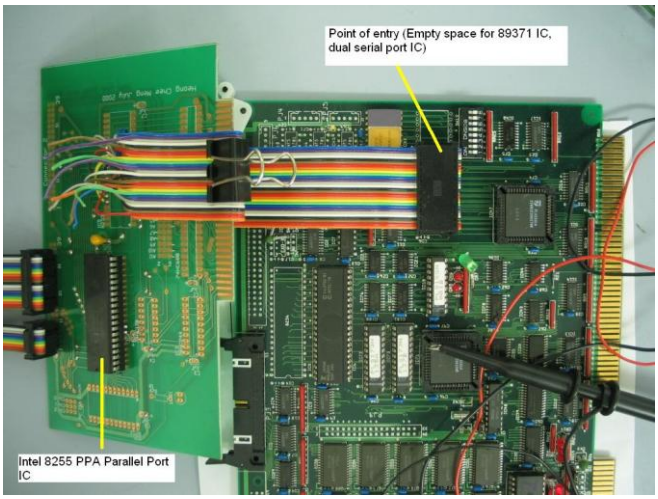


Figure 5: New entry point via the IO-mapped socket for MB89371 dual serial UART IC. This requires firmware change in the controller.

Even when the ICs are not available, the PCB is often only left unpopulated and it is often sufficient to install the correct IC (Figures 6 and 7). In these cases, since a communications IC does not have direct access to RAM, we need a small amount of firmware to be executed by the target CPU to perform the necessary memory fetches. This involves modification of the target firmware, so to minimize its impact on the rest of the code; it is preferable to add new code in the form of an interrupt service routine. Very often, existing footprints have already-allocated dedicated Interrupt Lines, and they can be utilized by intercepting the interrupt vectors involved.

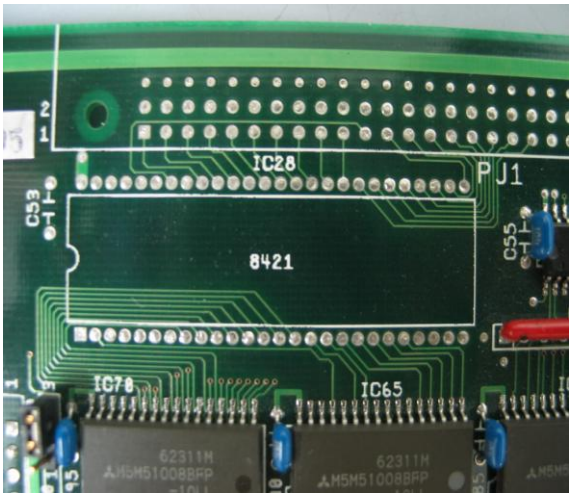


Figure 6: Alternative entry-point via MB8421 Dual-port RAM IC

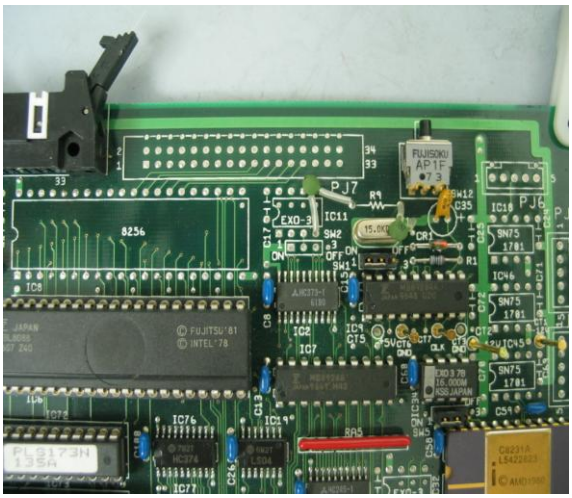


Figure 7: Entry point via MB8256 UART IC. In many ways the ideal entry point. This is the original developer's IO port. Replacing all 5 ICs results in a working serial channel.

The Linux Communications Controller then activates the new code by signaling to the Emulator microcontroller which then triggers off a hardware interrupt in the target system.

Whenever target system firmware is modified some reverse-assembly of the firmware in unavoidable. This is to gain sufficient insight into its operation, in case we interfere with its interrupt handling (which in turn affects its kernel scheduling) and also in case the target system has segmented (or paged) memory. In the worst case, the recipe data accessed this way may intermittently belong to a different, invalid memory page. We will need to figure out the kernel interrupt-handling routines in use and use them in preference. Luckily, most memory management systems use general-purpose ICs for this purpose, and are well-documented, but we have encountered some custom ASICs that needed to be probed (Figure 8).

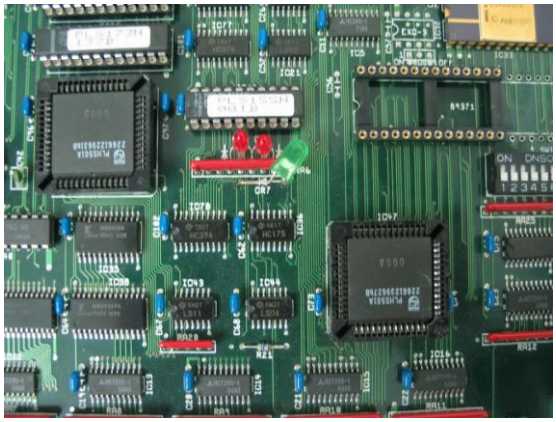


Figure 8: Square ICs are PLDs implementing custom memory controller logic. This has to be probed. Top right: note only a portion of the MB89371 socket is used.

Because of the need for reverse-assembly of target system machine-code, a RAM Emulator is very much the first choice.

Irrespective of devices implemented, the Linux Communications Controller hides these implementation details from the SECS/GEM Server. In addition Linux boasts a very wide variety of cross-assemblers and cross-compilers, in many cases even allowing us to generate the new code in gcc. Where necessary, legacy assemblers and compilers can be run as Virtual Machines using VMWare or Qemu regardless of the original operating system used.

3. RAM Emulator – Utilization

Among the functions for the RAM Emulator are recipe uploading, recipe downloading, auto recipe selection, retrieval of critical process parameters (online control), and remote start and stop of the machine.

There are basically two main reasons why we chose to develop RAM Emulator:

a) High cost of implementing SECS/GEM
b) Majority of old machines does not support SECS/GEM or does not support communication

For new machine there is a high chance that there is a SECS/GEM capability, but the drawback will be the cost of having it. It seems that the equipment manufacturers always charge very high for this item. However in some rare cases we might be able to negotiate on it, but this only applies to some equipment manufacturers and not all.

In some of the assembly sites, the majority of the machines consist of the older types. Whereby these machines either do not have SECS/GEM capability or cannot communicate at all. Since they contribute the bulk of the machines in the factory, we will need to address this issue. These machines are also the major contributor of the revenue for the factory, so we need to ensure these machines do not have any issues with the recipe (no mis-processing).

RAM Emulator	
Recipe Upload	√
Recipe Download	√
Remote Start	√
Remote Stop	√
Auto Recipe Selection	√
Auto Data Collection	√

Table I: Summary of RAM Emulator Functions

Function 1: RAM Emulator – Recipe Uploading

Recipe uploading is a process of putting the recipe into a centralized system(recipe server) from the machine via the host(RAM Emulator). Some machines have the capability of sharing one recipe to multiple machines(golden recipe concept). Others need to have a one to one relationship, one machine will have its own folder to store the recipes, and no sharing is permitted.

This process is needed in order to perform a correct recipe downloading. If there is no recipe that has been uploaded, then it is imppossible to perform a recipe download.

The upload process can be done from a graphical user interface (GUI) which is can be a web-based client, or it can be from the machine itself. If is is done via the GUI, the user just need to go to the specific website, select the machine and recipe that is intended to upload and with a click of a button, the recipe will be uploaded from the machine to the recipe server(this method is useful if the user is in the office). When the user is inside the production floor, they can simply go to the machine, and select the recipe that they want to upload, and click a button, so that the process is started.

Since the extent of the recipe area in memory is known, and often contiguous, we can download the entire recipe (often as little as 1KB) in a few milliseconds. The chain of communication can be thus: Recipe Server sends a command via TCP/IP or RS232C to the Linux Communications Controller. A dedicated Linux C program picks up the command and translates it to a stream of **bits** in JTAG 1174 format and directs to the Linux ppdev printer port driver, which causes the data stream to appear at the RAM Emulator's microcontroller. The latter reads the recipe data directly from its Dual-Port RAM and serializes it to JTAG-1174 which appears as a large bit-stream at the PC printer port. The receiving ppdev driver and C program de-serializes it into bytes and the resulting recipe file is sent to the Server (either using SECS/GEM, ftp, Openssh or Samba).

Function 2: RAM Emulator – Recipe Downloading

Recipe downloading is a process of transferring a recipe from the recipe server to the machine via RAM Emulator.

With recipe downloading we can ensure that the correct recipe is being used on the machine (no mis-processing).

This process must be initiated from a web-based client. We do not allow this from the machine, as if it is from the machine, the user will need to key in the recipe or perform a manual selection, and this will eventually result in selecting the wrong recipe. If it is through a web-based client, then it will be via scanning of a lot number. For this case we have decided to use a PDA. The PDA have a built in barcode scanner, that will facilitate the scanning of the lot number. Another reason why we chose the PDA is also because it is very mobile. In a line whereby there are a lot of machines, the user/operator can carry it to any of the machine and do the scanning there. By being at the physical location of the machine, we can make sure that the correct recipe is being downloaded for that machine, this will eliminate downloading of recipe to the wrong machine. Each lot will have a corresponding recipe. This being configured inside the database, whereby there is a device to recipe mapping. Each lot is associated with a particular device, as long as that device is inside the database, then the recipe will be available.

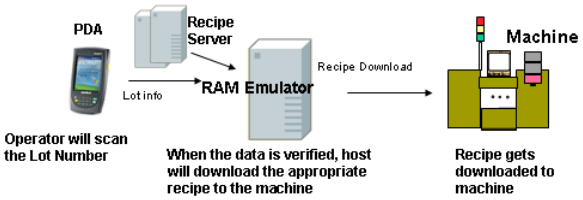


Figure 9: Recipe Download Architecture

Function 3: RAM Emulator – Auto Recipe Selection

In normal circumstances the downloading of a recipe will be to the hard disk. This is only the downloading portion, and the machine itself still have not load the recipe to its memory. As long as the loading of recipe to the memory is not done, then the actual recipe will always stay at the hard disk and will not be used, until the user manually loads the recipe. Auto recipe solution will solve this problem. After downloading, RAM Emulator will automatically loads up the recipe to the machine. In addition to that also, RAM Emulator will delete the rest of the recipes in the machine. This will ensure that only one recipe will exist in the machine at any one time, this is to prevent user from manually selecting another recipe, if there is more then one recipe.

Function 4: RAM Emulator – Retrieval of Critical Process Parameters

Aside from the recipe portion, the RAM Emulator is also able to retrieve critical process parameters. The data are being fed automatically from the machine via the RAM Emulator. This is part of the automatice data collection concept. An example of the critical process parameters are:

a) Bond Temperature
b) Bond Time Pad
c) Bond Power Pad

The list above is just an example of some of the critical process parameters, it can of course be more then that. The main reason why we require this is to ensure that the parameters are within the limits, and the machine is running with the correct parameters limit.

The process parameters can be retrieved by either user trigger or it can be a scheduled auto data collection. In the case whereby when it is triggered/collected by the user, it can be viewed first by the user before submitting it. If there is any abnormality on the data, then the user can take the necessary action. If in the case the user did not catch the abnormality and the user submit the data, there will be an alarm triggered. With this alarm, the user will know that something is not right with the machine. This also ties in with the remote stop capability, whereby if the machine is running, and the data is out of control, it will automatically be stopped. Another method of collection is via a scheduled collection (shifttly, daily, etc), whereby no user intervention is required. This method is less visual, whereby the user will not know what happened or seethe data. However, the mechanism on remote stop will still apply to this case if the data is out of the limit. A message box will be prompted on the machine to indicate the machine stopped because of limit violation.

Since this is automatic data collection, it will indirectly improve the productivity of the operators, as they no longer needs to manually look at the machine to figure out what is the value for those critical parameters.

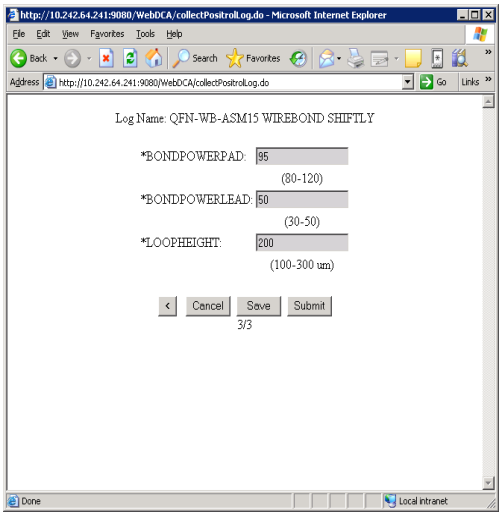


Figure 90: Auto data collection diagram.

Function 5: RAM Emulator – Remote Start

Remote start is whereby the machine will start the bonding when received a command from RAM Emulator. This will be done after the auto recipe selection is triggered. Due to the manual handling of materials on some of the assembly semiconductor machines this feature might not be that useful. It can also do some harm, if the user is adjusting the leadframe, and all the sudden the machine starts the bonding, this will be a disaster. However, for some of the machines that do not require manual handling, then this feature will be very useful.

It can be a major productivity improvement. In normal circumstances, the user will need to manually start each of the machine, but right now it is being started automatically, thus can saved a lot of time.

In response to server remote start command, the Linux Communications Controller starts the machine without reference to the RAM Emulator. It does this via digital output signal from its printer port (additional printer ports are easily added via USB adapters), which can then activate a dedicated ‘Start’ signal in the machines IO, or the ‘Start’ button in the machine keypad (via add-on relay board) or via one of the machine interlock sensors.

Function 6: RAM Emulator – Remote Stop

The Remote Stop feature is whereby the RAM Emulator will stopped the machine that is currently running. This is important because it can help to reduce scrap and to pre-alert some process related issues.

The remote stop can happen when the machine is running with some out of control parameters, then the machine can be stopped automatically. These out of control parameters can be detected either during the scheduled data collection by the user or the auto data collection (which runs in the background).

4. Security

Data security is a major issue, given the use of both WiFi and TCP/IP. The Linux Communications Controller currently uses WPA Encryption and Open Secure Shell (Openssh) for all adaptive routing scripts and programs.

Our WPA Encryption uses periodically renewed EAP keys. In addition Openssh is set up with DSA encryption with public/private key pairs. This has the effect of an ad-hoc VPN with good security. It also gave us much fewer points of intrusion (WiFi driver, Openssh, Linux kernel).

5. Beyond the SECS/GEM Retrofit

Taken to its logical conclusion, a target system having all its RAM and EPROM is replaced by RAM Emulator modules will be laid open to SECS/GEM. With WiFi-enabled RAM Emulator, this makes a pervasive or ubiquitous network ‘cloud’, where access is always available to CIM via SECS/GEM. This is a great advantage in ON Semi where manufacturing flexibility is the norm, and machines can be very quickly converted to run many different devices, moved to other locations or linked up for in-line operation. This allows continuity of CIM tracking and monitoring while maintaining manufacturing flexibility.

When fitted with a UPS (or when a laptop is used as Communications Controller) this lets us track machines even when they have been powered-down for conversion. The 'cloud' is thought of as maintaining SECS/GEM availability throughout machine changes as if the data has been incorporated into the factory physical infrastructure, much like the electric power socket or the pneumatic air supply or air-conditioning.

6. Conclusions

Recipe automation is very important in today's semiconductor manufacturing environment. Without it, we will encounter issues such as:

a) Mis-processing
b) Scrap
c) High cost of manufacturing – due to scrap
d) Labor intensive – due to a lot of manual handling on the machine side on the recipe portion, and also on the data collection

However with RAM Emulator we believe all the above issues can be solved. Besides that the cost of RAM Emulator is also far cheaper then buying a new machine, this is one of the main reasons we are going with RAM Emulator. A typical machine today (bundled with SECS/GEM) easily cost around 50 – 60k, while the cost of RAM Emulator is only 1% of it.

Acknowledgements

The authors would like to thank S.Kumar, Vice President and General Manager of ON Semiconductor Seremban, it is his vision that we strive to realize: that of the Lights-Out Factory. Special thanks to all the team members in Small Outline Surface Mount (SOSM) operation and CIM department (GT Chan, KY Puay, SK Teo) for their continuous support. We also appreciate helps and commitment from those involve either directly or indirecty in this project.

References

1. Semiconductor Equipment and Materials International, Book of SEMI Standards 2003.
2. CE Tan, KY Puay and S. Amin, "Methods to Achieve Zero Human Error in Semiconductor Manufacturing," *8th Electronic Packaging and Technology Conf*, Singapore, Dec. 2006
3. Simon Johnson, "EPROM Emulator For Selectively Simulating A Variety of Different Paging EPROMS in a Test Circuit", United States Patent No: 5,003,507
4. Mick Bauer, "The 101 Uses of Openssh", Linux Journal Feb 1, 2001