UNIVERSITA' DEGLI STUDI DI PADOVA

FACOLTA' DI INGEGNERIA
CORSO DI LAUREA IN INGEGNERIA DELLE TELECOMUNICAZIONI

ANALISI QUALITA' DEL SOFTWARE PER IL TEST DI MEMORIE NON VOLATILI: CASI DI STUDIO

SOFTWARE QUALITY ANALISYS FOR NOT VOLATILE MEMORIES TESTING: CASE STUDIES

Laureando: Rizzi Alberto

Relatore: prof. De Poli Giovanni
Tutor aziendale: ing. Bonacina Francesco

Anno accademico 2011/2012

INDICE

LISTA DELLE FIGURE

1 INTRODUZIONE

La presente tesi si basa su un lavoro di tirocinio svolto presso la sede di Padova di Infineon Technologies.
Infineon è un'azienda produttrice di semiconduttori, con sede principale a Monaco di Baviera, fondata nel 1999 dallo spin-off di Siemens AG. Il Design Center di Padova, fondato nel 2001 e che conta circa un centinaio di ingegneri specializzati in sviluppo e supporti dei prodotti automotive, ospita un team di *Product Test Engineering (PTE)* dedicato allo sviluppo di metodologie e di software per il testing (TestWare) di memorie non volatili (NVM memories: Flash, EEPROM) incorporate nei microcontrollori dedicati al mercato automobilistico. In particolare il focus del team copre dall'analisi del silicio al controllo dell'efficienza delle funzionalità del dispositivo garantite al cliente finale. Il processo si suddivide principalmente in quattro fasi:

- **Analisi**: quest'attività parte dalla necessità di valutare il dispositivo in determinate condizioni per capirne le performance o generare nuove specifiche per i dispositivi futuri.
- **Validazione**: è il processo che verifica che il dispositivo rispetti i requisiti presenti nei datasheet e le necessità dei clienti.
- **Caratterizzazione**: il dispositivo viene fatto funzionare in diverse condizioni. Viene variata sia la temperatura che la tensione in ingresso e si osserva come reagisce il microcontrollore.
- **TestWare**: gli ingegneri che fanno parte del gruppo PTE hanno definito una lista di test da eseguire sul dispositivo per la sua produzione.

Sviluppare dei test per una memoria Flash è una parte molto costosa del processo di produzione di microcontrollori: è richiesto un tempo di test ridotto e una copertura del maggior numero possibile di funzionalità. Il gruppo PTE di Infineon ha lo scopo di ottimizzare la produzione di test e per questo sono coinvolti molti ingegneri: il gruppo che concepisce sofisticati algoritmi e i relativi flussi, il team che sviluppa i programmi da eseguirsi su *Automatic Test Equipments (ATE)* collocati nei siti produttivi, gli ingegneri di prodotto che necessitano di tool e software per ottimizzare la caratterizzazione dei microcontrollori *(Device Under Test- DUT)* e per eseguire l'analisi dei fallimenti.
Nel 2002 il gruppo PTE ha scelto di usare il *System on Chip(SoC)*, una metodologia di test che utilizza le risorse di calcolo e di memoria interne al dispositivo stesso. Questa tecnica consiste nell'utilizzare le risorse dei DUT, per testarne le memorie dopo aver eseguito alcuni test logici sui transistor dei dispositivi. I test sulla Flash si prestano bene all'utilizzo di questa metodologia in quanto richiedono un'elevata velocità di accesso ai dati. Questo risultato è ottenuto grazie al caricamento ed all'esecuzione in SRAM di un firmware di test dedicato, detto TestWare. Il TestWare soddisfa i requisiti di test sulla

memoria Flash. Le risorse disponibili dei DUT sono sufficienti per supportare gli algoritmi complessi e garantire una veloce esecuzione.
Il gruppo PTE di Padova è il gestore ufficiale del TestWare e, nel 2007, il team ha sviluppato un concetto per standardizzare la creazione e l'esecuzione dei test, rispondendo al requisito di essere supportata da differenti ATE, e avere un codice sufficientemente agile da caricare nei diversi dispositivi. Ciò ha portato alla nascita del *Functional Test Operating System (FTOS)*. L'FTOS è un livello software di interfaccia tra gli ATE e i DUT, responsabile della gestione delle attività e delle risorse.
FTOS è caricato nella memoria SRAM all'inizio del flusso di test e supporta l'esecuzione di programmi della famiglia *Test Applications (TAs)* che contiene funzioni e algoritmi complessi dedicati alla verifica e validazione della Flash: FTOS fornisce la struttura di esecuzione, mentre le TAs provvedono agli algoritmi di test.
Nuove versioni di Test Applications vengono rilasciate molto più spesso rispetto alle versioni dell'FTOS, che deve variare solamente nel caso di modifiche hardware di ATE e DUT o per l'introduzione di un nuovo microcontrollore.
In questo contesto l'obiettivo di lavoro del mio tirocinio, all'interno del team è stato quello di creare da zero (concept → release) dei programmi in grado di validare il corretto funzionamento di FTOS in tutte le sue caratteristiche.

1.1 Software testing

Il software testing viene utilizzato per controllare la qualità del prodotto, trovare eventuali bachi o funzionalità non coperte quando invece è previsto che lo siano: in questo modo si accresce la robustezza del prodotto.
Le tecniche di test utilizzate non sono effettuate al solo scopo di trovare dei bachi all'interno del software, ma anche con l'intento di verificare e certificare che il software risponda ai requisiti richiesti dagli sviluppatori e dai committenti, anche in diverse condizioni sia climatiche che elettriche.
Il testing del software può essere eseguito sia in parallelo al processo di sviluppo del codice, sia a processo ultimato, a seconda di quando il test viene effettuato si utilizzano metodologie differenti.
In molti modelli tradizionali la quasi totalità dei test viene eseguita dopo che il processo di sviluppo del codice è già stato completato, allungando in questo modo i tempi di accertamento della qualità. Nel momento in cui gli sviluppatori ritengono di aver terminato la scrittura del codice, rilasciano una versione, denominata beta version, che deve passare a chi si occupa di analisi della qualità.
Effettuando dei test sul software non si avrà mai la certezza di aver trovato tutti i bachi, ma si può cercare di andare a testare il maggior numero di funzionalità possibile. Infatti, la quantità di software di test dipende dalla qualità che si vuole dare al prodotto e da chi

dovrà utilizzarlo: ad esempio il software di agenzie governative deve avere meno probabilità di errore di un normale video game.

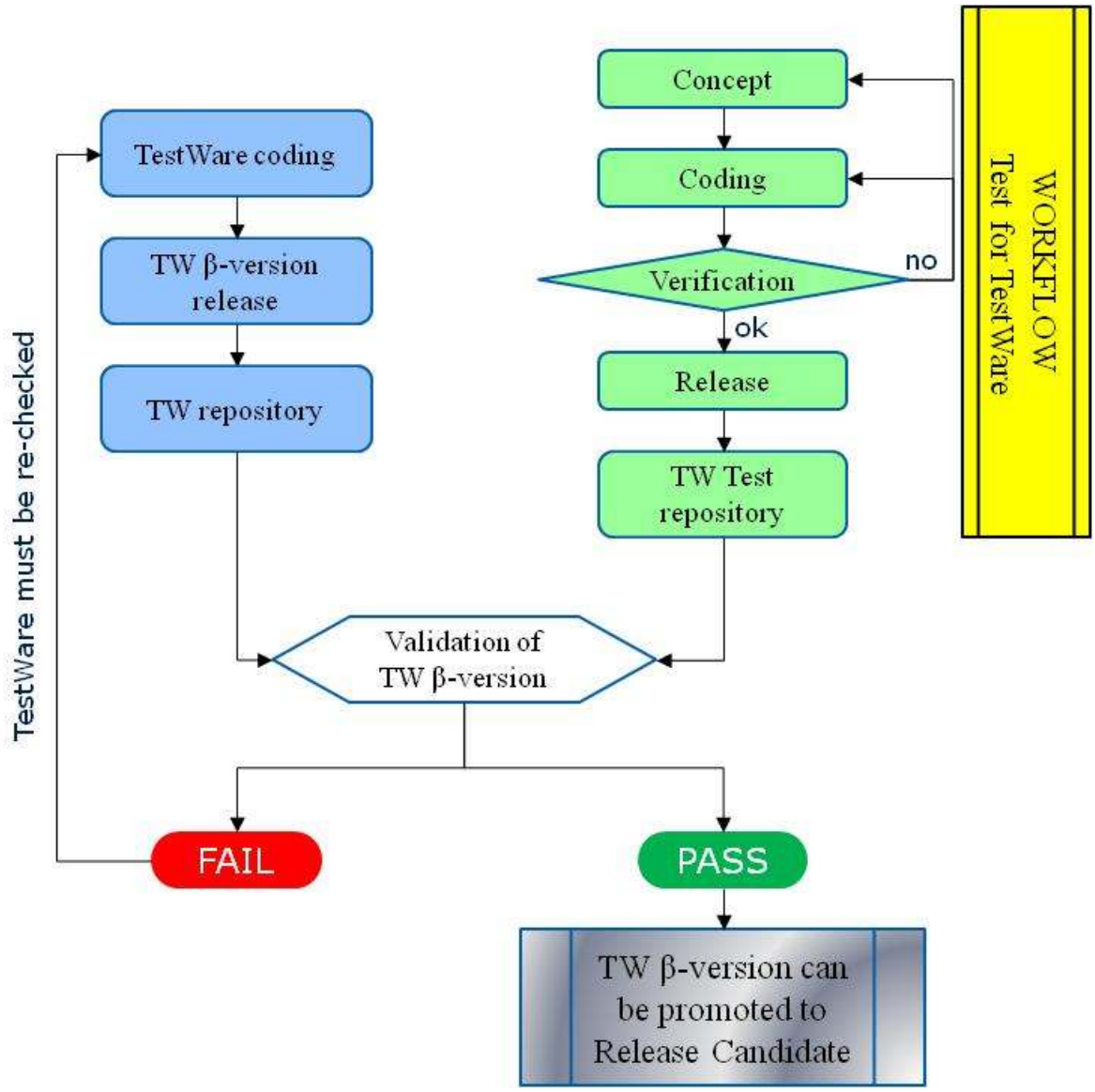


Figura 1: Diagramma di flusso per la verifica del software.

Diversi studi hanno dimostrato come gli errori e i bachi presenti nel software siano più costosi, sia in termini economici che di risorse, rispetto al costo di realizzazione del software di test.

Solitamente la fase del processo di sviluppo del software nella quale viene trovato un fallimento, determina il costo che questo implicherà per lo sviluppatore: prima viene trovato e meno costoso è risolverlo.

In generale, un software di test non può stabilire che un determinato prodotto funzioni in tutte le condizioni, ma può trovare in quali condizioni questo prodotto non funzionerà.

Esistono numerose classificazioni dei test del software: principalmente questi si possono suddividere in test funzionali o non funzionali. I test funzionali fanno riferimento alle specifiche proprietà del prodotto, solitamente descritte nella documentazione, e vanno quindi a testare se effettivamente la funzionalità specifica agisce come descritto.
I test non funzionali, invece, controllano altri aspetti del prodotto che possono variare dalla sicurezza alle performance a diverse condizioni.
Alcune volte i programmatori sono a conoscenza di alcuni difetti del loro software, questi però non devono essere considerati dei fallimenti, ma solamente dei limiti del prodotto. Il software di test può servire anche a prendere atto dei limiti del prodotto, in modo che si possa decidere se investire risorse nell'eliminazione del limite o risparmiare le forze e modificare le caratteristiche nella documentazione. Alcuni difetti del prodotto, per il cambiamento di diverse condizioni, possono divenire dei fallimenti.
Una delle cause maggiori di fallimento è la compatibilità con il sistema, nel senso che un software può non funzionare nella versione più aggiornata di un sistema in cui prima funzionava. D'altro canto può accadere anche l'opposto e utilizzare il software su versioni precedenti del sistema può far incorrere in fallimenti.
Uno dei problemi più importanti del software testing è coprire tutte le condizioni e le diverse combinazioni di input che il prodotto da testare può ricevere.
Ci sono diversi approcci alla metodologia di test, comunemente si parla di black box, white box e grey box.

- Si parla di black box quando il programmatore non è a conoscenza del contenuto del software da testare. Il tester, basandosi sulle specifiche del prodotto, inserisce alcuni input e deve aspettarsi degli output certi, se ciò non avviene ha trovato un fallimento.
- Per white box si intende quando il tester ha a disposizione il codice da verificare, le strutture interne e gli algoritmi. Questa metodologia di test viene anche utilizzata quando si vuole valutare la completezza di un test realizzato con il metodo Black box. Così facendo si può stabilire una percentuale di copertura del test. Utilizzando il metodo del Black box vengono trovati molti bachi che con il white box non vengono rilevati. D'altro canto però sono necessari molti più test per verificare una singola funzionalità.
- Il grey-box è quando viene a crearsi una situazione ibrida in cui il tester è a conoscenza delle strutture e degli algoritmi interni del prodotto, e fornisce alcuni input necessari a verificare che determinate funzionalità avvengano.

Una tipologia di test che si può considerare un black box è il regression testing. Questo tipo di test si focalizza sul trovare dei bachi nel prodotto principale quando viene introdotto un cambiamento. Principalmente il software di regression serve per trovare errori non presenti nelle vecchie versioni del prodotto e che emergono quando viene effettuata una modifica, magari anche in parti di programma che non dovrebbero essere intaccate dal problema. Anche la correzione di un baco può indurre errori in altre parti

del codice che prima non erano presenti. Un metodo abbastanza comune di verifica del software è eseguire i test sul prodotto prima della modifica (tutti i test devono passare), apportare la modifica e poi rieseguire i test di regression. A questo punto, se si generano dei fallimenti, è chiaro che sono state le ultime modifiche apportate a inserire nuove instabilità: è quindi necessario analizzare il codice del TestWare per risolvere problema. Il problema al 90% è sito nel TestWare poiché si suppone che tutti i test di regression, prima di essere rilasciati, siano bug-free e che siano compatibili con tutti i microcontrollori. Spesso accade che dei bachi o dei fallimenti riemergano uguali nel tempo. Questo avviene perchè può essere rilasciata una nuova versione del codice dimenticandosi del problema capitato in precedenza. A volte può succedere che un baco sia eliminato solamente a determinate condizioni e, nel momento in cui vengono a mancare queste condizioni, il fallimento si ripresenta. E' necessario quindi eliminare il baco in modo più generale.

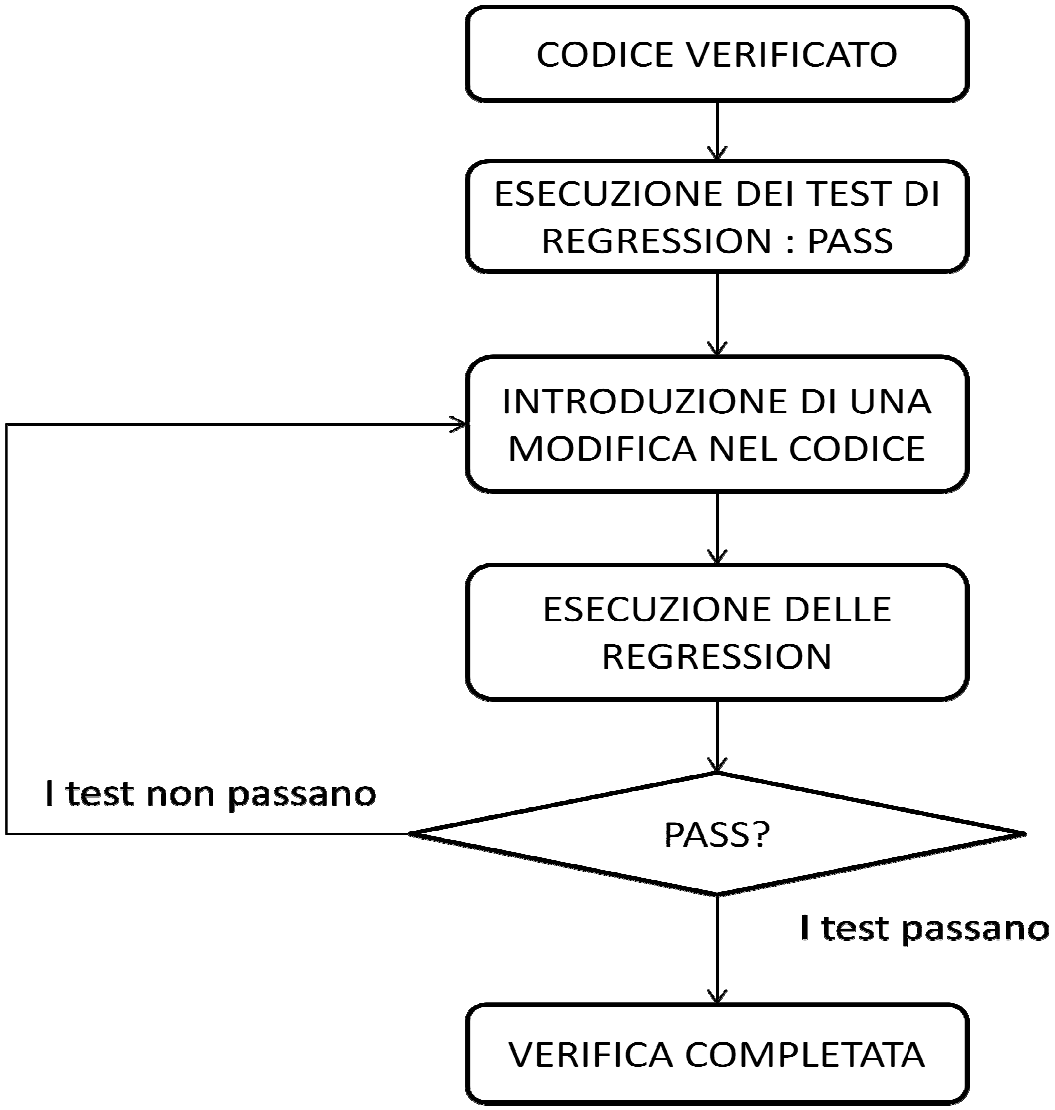


Figura 2: Diagramma di flusso dei test di regression.

E' buona norma, quando viene trovato un baco, creare un test di regression dedicato che faccia emergere il fallimento. In questo modo, ogni volta che verrà rilasciata una nuova versione di codice, sarà eseguito un test apposito che verificherà l'assenza del fallimento specifico.
Nella metodologia di lavoro, come accade in Infineon, i test di regression sono eseguiti da un tool automatico (RegGAE© - **Reg**ression **G**ear in **A**utomated **E**nvironment) che testa i DUT ogni notte e, in caso di fallimento invia un messaggio al responsabile del prodotto.
I test di regression possono essere utilizzati, oltre che per trovare bachi nel sistema, per studiare le performance e la qualità degli output. Le regression devono coprire tutte le funzionalità del dispositivo e devono generare il maggior numero possibile di testcase in modo da verificare il funzionamento del prodotto in un numero considerevole di condizioni.

1.2 Scopo del tirocinio

Il team MC di Padova da alcuni anni produce ed esegue dei test di Regression sulle Test Applications di diversi prodotti. Lo scopo del mio tirocinio è stato creare un insieme di test di Regression per il *Functional Test Operating System (FTOS*).
Al mio arrivo in Infineon ho concordato con l'ingegner Francesco Bonacina e Antonio Fin, i possibili test di regression da realizzare. Questi test coprono solo una parte di un insieme più ampio di funzionalità realizzate da FTOS: dall'avvio del sistema, passando per la generazione di errori, fino alla gestione dell'esecuzione di comandi.
All'interno del team MC il codice del TestWare è scritto da più persone e, nonostante siano state definite delle linee guida per la sua stesura, ognuno scrive in modo personale: è quindi importante che sia presente un filtro comune a tutti i listati prodotti e che sia indipendente da chi ha scritto il codice stesso.
Effettuare dei test di Regression su FTOS rende il prodotto robusto, nel senso che viene ridotta la probabilità di farsi sfuggire un errore quando viene generata una modifica sul codice. In questo modo vengono aumentate l'affidabilità del prodotto e la qualità stessa.
All'inizio del periodo di tirocinio sono state decise alcune particolari funzionalità da testare, sia in base alla priorità che queste avevano, sia alla difficoltà e alle nozioni necessarie per poter creare i test di regression opportuni. Le regression create, dopo averne testato il corretto funzionamento, devono essere caricate in RegGAE©, un sistema sviluppato da Infineon, che tutte le notti esegue sui microcontrollori tutti i test di regression disponibili nella repository.
Per iniziare il lavoro sono state scelte tre funzionalità ad alta priorità:

1. Il corretto funzionamento del comando set del PLL;
2. Il corretto funzionamento del comando Check CRC;
3. Il corretto funzionamento del comando Delay.

2 ORGANIZZAZIONE DEL LAVORO

2.1 HW e SW tools

I programmi di regression sono stati realizzati in Perl script, un linguaggio di programmazione di utilizzo generale, sviluppato per gestire testo e ora utilizzato per diversi scopi tra cui: lo sviluppo di pagine Web, la gestione di sistemi, programmazione di reti e molto altro. Il Perl è stato concepito con l'intento di essere pratico, efficiente e facile da utilizzare, ma anche elegante e senza elementi superflui. Il Perl è caratterizzato da una sintassi molto flessibile e uno stesso algoritmo può essere implementato in modi diversi producendo lo stesso risultato. Questo linguaggio di programmazione consente di processare file e manipolare testi molto semplicemente e, quindi, è indicato nei casi in cui si ha un'interazione diretta con il sistema operativo. Il fatto che il Perl non richieda di essere compilato prima dell'esecuzione rende molto semplice la realizzazione e la correzione dei programmi. Il Perl non è un linguaggio di programmazione velocissimo, ma consente comunque di fare degli script piuttosto complessi.

Figura 3: Marchio di Perl

Durante il tirocinio, come editor di testo e strumento di debug ho utilizzato Eclipse: un ambiente di sviluppo del software affidabile e collaudato utile per progettare e sviluppare semplicemente programmi per utilizzo personale e commerciale. Il software di Eclipse è open source, dunque è gratuito e scaricabile in piena libertà. Lo sviluppo e il miglioramento di Eclipse sono curati sia da singoli programmatori per passione, sia da grandi aziende quali HP, IBM ed Eriksson.

Figura 4: Marchio di Eclipse.

Il progetto di Eclipse è basato sui plug-in, tramite i quali, il software può essere configurato al meglio per le proprie esigenze. Nel mio caso ho installato un paio di plug-in per debuggare gli script Perl e connettermi al dispositivo.

Il tool di analisi del microcontrollore utilizzato da Infineon è Jazz, uno strumento software creato ad hoc per l'analisi e il testing delle memorie non volatili.

Questo strumento è essenziale per interfacciare il DUT, ma può anche essere utilizzato per controllare diversi strumenti di misura tra cui multimetri e alimentatori attraverso numerosi protocolli tra cui TCP/IP, GPIB e RS232.

Il tool di Jazz è basato sull'ambiente di un PC, ma può essere connesso e sincronizzato con altri PC per testare più dispositivi in parallelo.
Jazz è basato sulla *Graphical User Interface (GUI)*, facile da usare per l'utente e riduce le probabilità di errori umani.
Jazz si propone di essere:

- standard, integrato;
- indipendente dai prodotti (8, 16, 32 bit);
- un tool di analisi e caratterizzazione;
- un sistema che rende automatiche le attività di validazione e caratterizzazione;
- un tool che supporta differenti flussi per differenti analisi;
- facile da utilizzare;
- una struttura che carica i risultati per le differenti operazioni.

Utilizzando il tool di Jazz è possibile interfacciare il personal computer con:

1. Il Chip: facendone l'inizializzazione e scaricando gli applicativi;
2. La Flash: in lettura, scrittura e cancellazione;
3. L'SRAM: in lettura, scrittura e cancellazione;
4. Alimentatore: dall'accensione e settaggio valori alla misura;
5. Oscilloscopio: dall'accensione alla misurazione;
6. Generatore di funzioni: dall'accensione alla forzatura di forme d'onda;
7. Thermostream (una macchina che tramite un getto d'aria permette di portare i dispositivi a diverse temperature per ricreare situazioni realistiche di stress termico per le memorie): dall'accensione al settaggio valori.

In Jazz è possibile rappresentare i dati attraverso modelli grafici che utilizzano diversi parametri: dal semplice numero di fallimenti alla caratterizzazione a diverse temperature.
Il tool di Jazz è in continuo sviluppo per rimanere aggiornato con le esigenze del team o per nuovi prodotti: una volta che una nuova versione di Jazz viene rilasciata, viene distribuita al team per una prima validazione.
Il sistema hardware utilizzato è composto essenzialmente da quattro apparecchi:

- Il personal computer;
- La mini wiggler;
- La board contenente il DUT;
- La scheda che stabilizza le tensioni.

Partendo dal PC, con un semplice attacco USB ci si collega ad una mini wiggler che attraverso un bus dati con protocollo JTAG è collegata alla board in cui è posto il chip da testare (DUT). Questa board è alimentata attraverso un alimentatore oppure per mezzo di una scheda dedicata che fornisce delle tensioni stabili.

La mini wiggler è uno strumento non molto costoso realizzato da Infineon per interfacciare le board al PC attraverso l'USB.
Nel mio caso la board è stata alimentata da un'apposita scheda, creata da un altro team di Infineon, che fornisce le tensioni necessarie ad alimentare il Chip. Nel caso in cui il chip dovesse rompersi o perdere la configurazione di base si può semplicemente sostituire.
Il chip può lavorare a diverse frequenze, ma è necessario mettere un quarzo o un oscillatore in modo da generare una frequenza di base che poi viene presa ed elaborata dal microcontrollore.

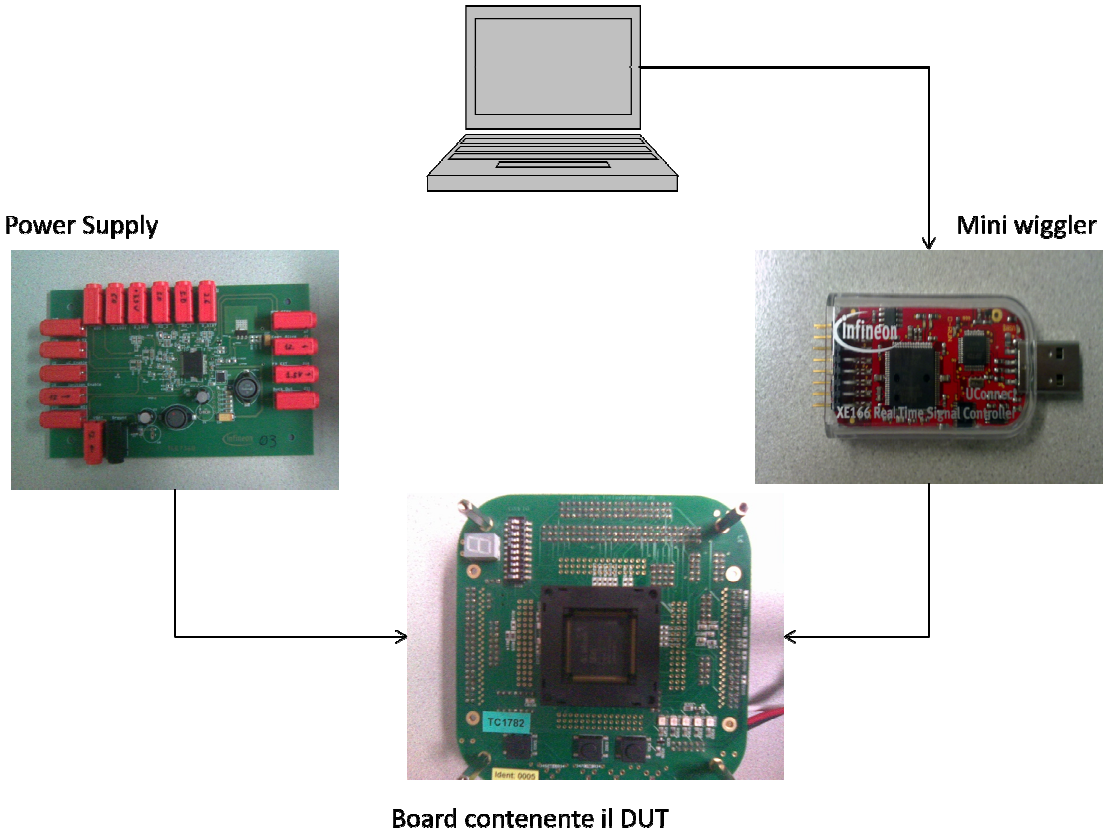
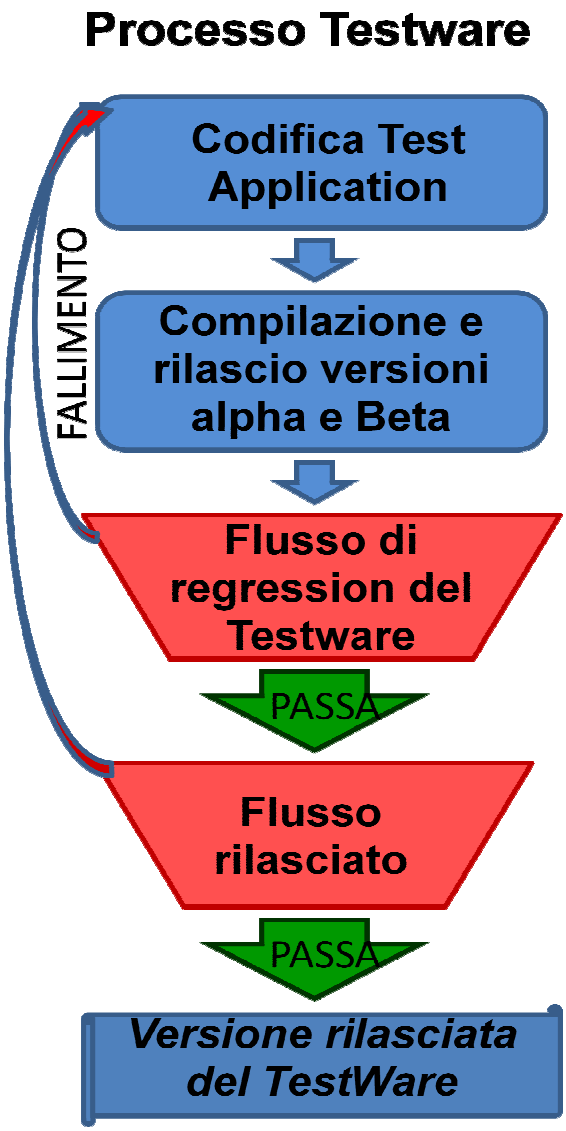


Figura 6: Configurazione Hardware del sistema

2.2 Metodologia

Attraverso il tool di Jazz è possibile creare collezioni di test per il singolo dispositivo che consentono di generare dei flussi da scaricare nella memoria del DUT.
Un flusso di test è un insieme di prove che consente di eseguire determinate analisi o caratterizzazioni e, nel caso sia necessario, può contenere anche più volte lo stesso test e deve restituire sempre come risultato un "PASS" o un "FAIL".
Un flusso di test è identificato da un nome e da una modalità di esecuzione nella quale i test vengono eseguiti in sequenza uno dopo l'altro. Ogni test può essere eseguito anche più volte prima di passare al successivo oppure, in alcuni casi particolari, un test può essere saltato. Per ogni test può essere scelto un risultato "aspettato" in modo tale da

terminare il flusso nel caso occorra un risultato diverso. Se il test è indipendente dal dispositivo, è possibile utilizzare lo stesso test per più dispositivi.
Inizialmente, ogni volta che un nuovo prodotto veniva realizzato, era necessario generare una notevole quantità di script in Perl. Per aggirare il problema è stata concepita e realizzata una libreria di comandi, detta PerlCore, che contiene moduli perl che possono essere riutilizzati per molti prodotti, esistenti o futuri. Questa libreria è stata realizzata focalizzandosi sulla collaborazione tra sviluppatori e utenti. Inoltre la libreria è stata implementata in modo totalmente indipendente dagli altri tool con i quali comunica tramite interfacce. Prima di rilasciare una nuova versione di PerlCore, essa viene utilizzata dai Beta tester che ne segnalano eventuali errori e problemi. Durante lo sviluppo delle librerie del PerlCore viene generata automaticamente la documentazione per gli utenti.



e del Testware

La catena di produzione del TestWare richiede un controllo qualità che verifichi il corretto funzionamento di ogni singolo step. La creazione dei Regression Test si è resa necessaria in quanto ogni volta che viene introdotta una nuova funzionalità nel TestWare, questa può compromettere la qualità e i risultati ottenuti da funzionalità precedenti. Per essere certi dell'elevata qualità del prodotto sarebbe necessario collaudare il TestWare in tutte le sue componenti ad ogni modifica. Questo collaudo di funzionalità già esistenti e già verificate è chiamato "collaudo di regressione" (Regression Testing) in quanto si vuole verificare se la qualità è regredita.
Se il product engineer pubblica una nuova versione di applicativo, deve prima eseguire i flussi esistenti che riguardano il prodotto e l'applicativo. Nel caso in cui avvenga un cambiamento di funzionalità o l'inserimento di nuove capacità del dispositivo, risulta necessario produrre nuove regression che vadano a coprire le nuove opzioni.

La creazione di una nuova regression inizia quando il Lead Engineer (L.E.) richiede un cambiamento, modificando così il flusso di lavoro. Il TestWare engineer, dopo aver discusso le modifiche con il L.E. propone un'implementazione, descritta nel documento ufficiale interno detto "Implementation spec", che se accettata passa alla fase di codifica e implementazione. In seguito c'è la fase di compilazione delle versioni alpha/beta che vengono sottoposte alle regression di prodotto e di applicativo. Nel caso sia stata inoltrata una richiesta di cambiamento, nei flussi di test saranno inseriti i test specifici.

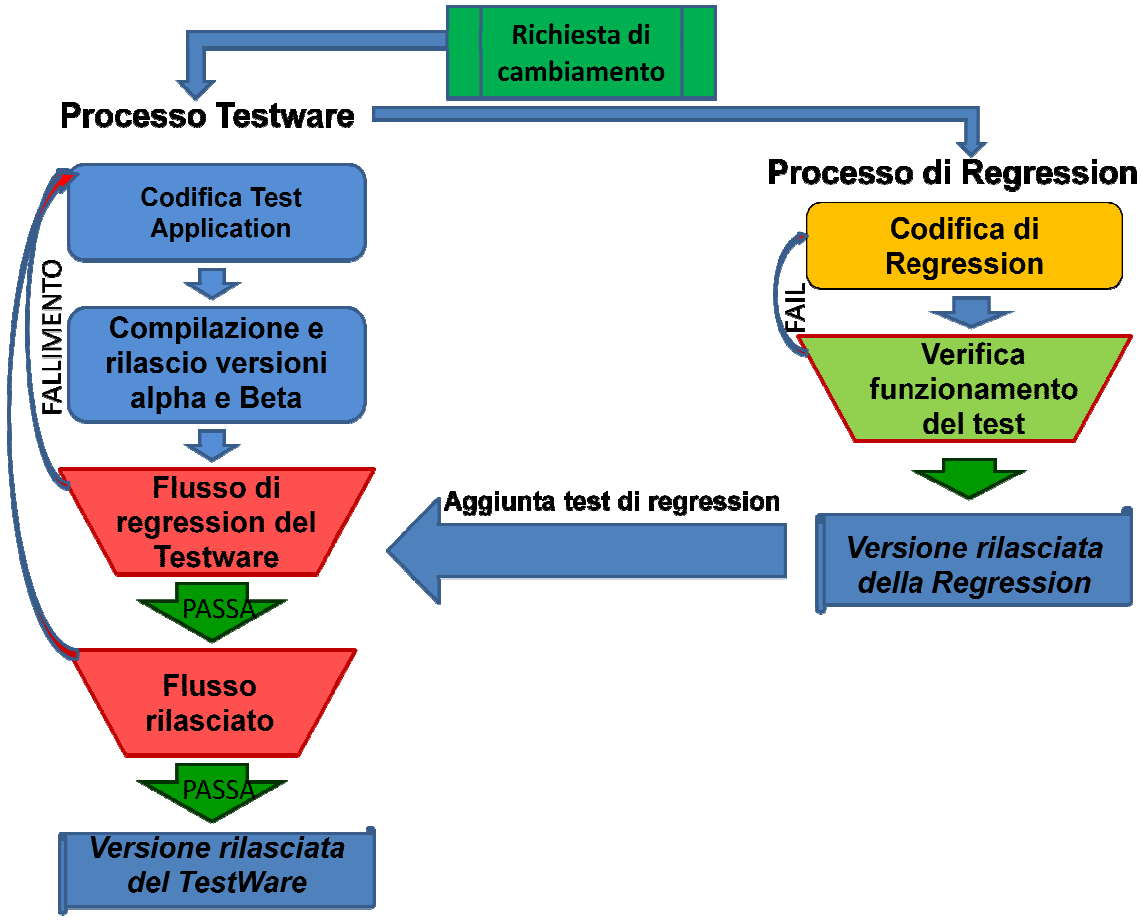


Figura 8: Modifica della sequenza del TestWare in seguito ad una richiesta di cambiamento

Per aiutare chi deve creare delle regression per la prima volta sono stati forniti due strumenti molto utili:

- Regression template: è uno script Perl dal quale poter iniziare l'implementazione del proprio test di regression, contiene alcune chiamate a funzioni standard, il caricamento delle librerie più usate oltre al codice di connessione e chiusura del programma.
- Regression Perl module: è un strato software che sta tra il PerlCore e lo script di regression che ha un doppio scopo: creare funzioni utilizzate dai TestWare engineer in diversi programmi e con diversi dispositivi da poter richiamare

all'occorrenza e rendere le regression indipendenti dal PerlCore (nel caso venga eseguita una modifica in una libreria, non è necessario andar a modificare tutte le regression ma solo il modulo Perl.

Uno script di regression, all'interno di Infineon, deve rispettare alcuni requisiti:

1. Indipendente dal prodotto:
   quando viene creato uno script in Perl, bisogna tenere a mente che deve poter essere eseguito su tutti i prodotti, o comunque su tutti quelli che prevedono la funzionalità che deve essere testata. Per questo motivo bisogna evitare di inserire indirizzi di memoria numerici, ma è necessario utilizzare alcuni file di testo, diversi da prodotto a prodotto, in cui sono presenti degli indirizzi di memoria e le relative etichette (`#define`). Questi file vengono caricati da Jazz che, a seconda del prodotto, va a generare un albero di etichette che vengono utilizzate dai diversi script. E' necessario prestare attenzione a non modificare i nomi dei define quando vengono messi sul mercato nuovi prodotti.
2. Minor uso possibile di subroutine:
   questo avviene grazie all'utilizzo del PerlCore che va a coprire moltissime funzionalità. Prima di andare a creare una subroutine conviene sempre guardare se questa non è già coperta da una libreria esistente.
3. Struttura di regression:
   per una più facile lettura da parte di tutti i componenti del team è necessario mantenere una struttura costante e condivisa che tenga conto di alcune linee fondamentali. Nella prima parte dello script sono presenti le variabili e le costanti da utilizzare, poi gli input, i risultati e gli output aspettati. Nella parte centrale si sviluppa l'algoritmo di regression seguito dalla sezione che gestisce i risultati che devono essere controllati con quelli aspettati. Infine è necessario stampare a monitor i risultati, per una più facile comprensione.

3 FASI DI LAVORO

Durante i primi giorni di tirocinio è stato deciso il mio work plan in modo da definire in modo chiaro il processo di lavoro con tempi e risultati richiesti.

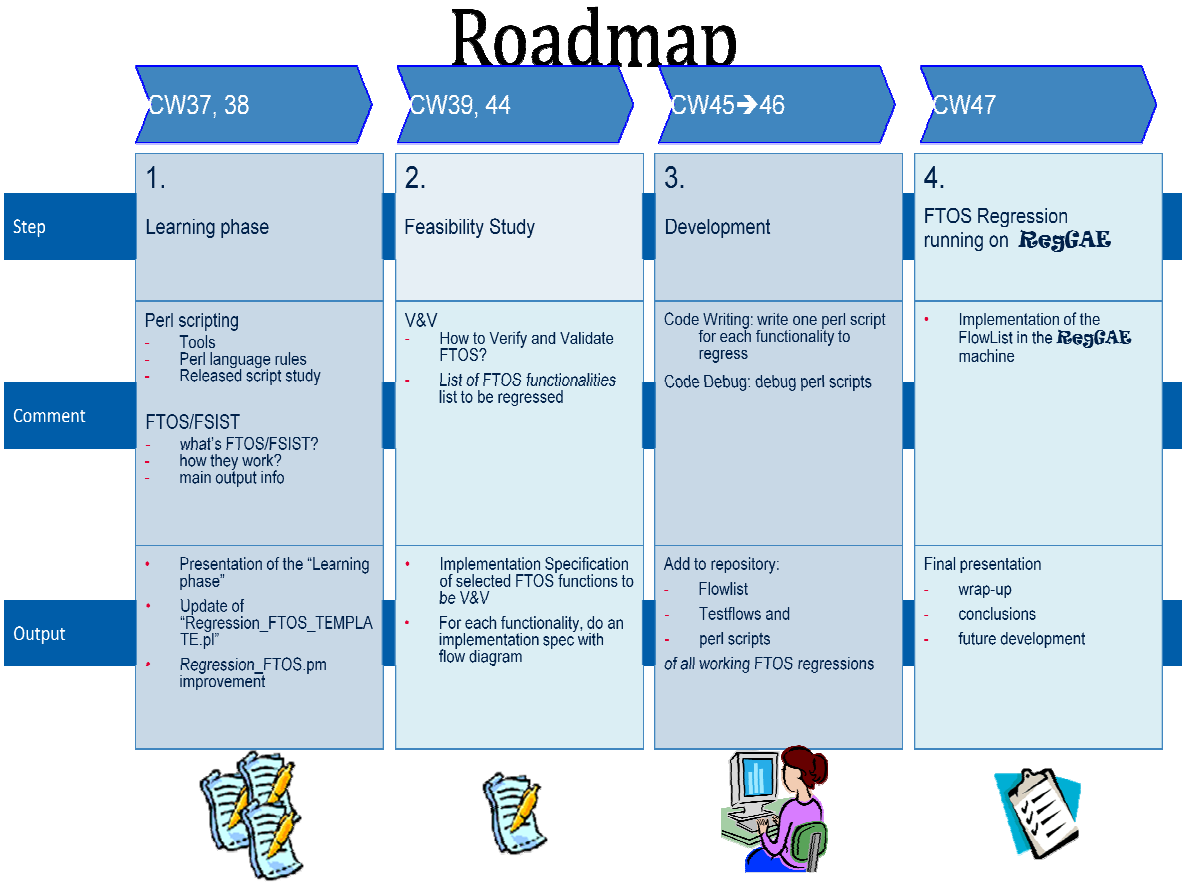


Figura 9: Roadmap suddivisa in settimane di lavoro

In questa roadmap, è stata stabilita una tabella di marcia in cui sono ben definite le quattro fasi di lavoro:

- Fase di studio;
- Studio di fattibilità;
- Sviluppo dei test di regression;
- Inserimento delle regression in RegGAE©.

La creazione di una Roadmap è stata molto utile per strutturare il lavoro, anche se i periodi delle varie fasi erano solamente indicativi, mi hanno dato l'idea dell'avanzamento del lavoro.

3.1 Fase di studio

Il lavoro, nei primi giorni, è consistito principalmente nella configurazione del computer e di tutti gli strumenti software e hardware da utilizzare: è stato un lavoro di configurazione per trovare il set-up operativo.
Durante la prima fase di studio, i temi analizzati si possono suddividere principalmente in due argomenti: il primo riguarda la scrittura del codice e il secondo il software su cui eseguire i programmi di regression.
Per la parte riguardante la scrittura del codice è stato necessario studiare il linguaggio di programmazione Perl e le sue strutture fondamentali, oltre che imparare ad utilizzare tutti gli strumenti necessari ad interfacciarsi con il dispositivo (Jazz ed Eclipse).
Per prendere confidenza con la programmazione Perl ho iniziato con la creazione di alcuni semplici programmi che utilizzavano le strutture fondamentali di qualsiasi linguaggio di programmazione: dall'if-then-else agli hash e gli array.
Nello stesso momento, con questi semplici algoritmi, ho iniziato ad accedere ad alcune locazioni di memoria dei *Device Under Test (DUT).*
Per aver accesso al dispositivo e programmare all'interno della sua memoria, è stato necessario creare un "flusso" di download apposito per il singolo dispositivo: in questo modo si vanno a caricare FTOS e Test Application in SRAM.
Dopo aver creato il flusso di download bisogna selezionare il dispositivo su cui andar ad eseguire i test ed avviare il download.

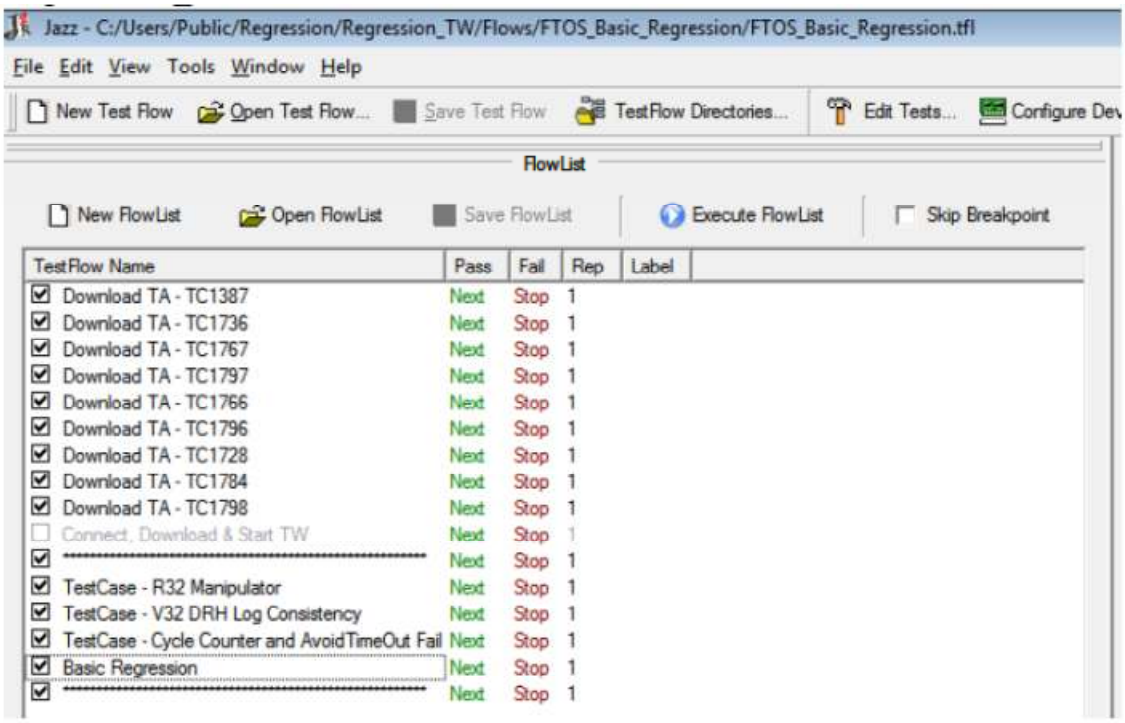


Figura 10: Esempio di flusso di download

A questo punto il dispositivo ha all'interno della memoria SRAM sia Test Application che FTOS ed è quindi pronto ad eseguire i comandi utili a testare la memoria flash.
Per realizzare i programmi di Regression è stato creato un Perl module che deve essere caricato all'interno del programma. Un Perl module è un insieme di funzioni e algoritmi che possono essere richiamate da altri programmi. Nel mio caso, ho utilizzato un Perl module apposito per le regression, che contiene le funzioni più utilizzate in questo tipo di programmi. All'interno di questo Perl module sono presenti anche funzioni che richiamano direttamente subroutine presenti nel PerlCore.
Ogni processo eseguito all'interno dell'azienda è accompagnato da una guida che ne spiega le finalità e i possibili utilizzi: grazie a queste guide, è più facile imparare ad utilizzare tutto il necessario per il lavoro.
Le regression devono essere indipendenti dal prodotto sul quale vengono utilizzate, anche se i dispositivi sono numerosi ed eseguono alcune funzionalità in maniera differente, le regression devono saper gestire la situazione e segnalare un errore solo nel caso di un eventuale baco o fallimento.
Dopo aver studiato il funzionamento di parti comuni a tutti i dispositivi, ho utilizzato l'*Internal Target Specification (ITS),* una guida specifica per ogni singolo prodotto che spiega in modo molto approfondito l'intero dispositivo. Nel mio caso ho utilizzato la guida del microcontrollore a 32 bit TC1784 (evidenziato nella figura sottostante).

Qui sotto è rappresentata la famiglia di microcontrollori a 32 bit di Infineon suddivise per performance ed anno di messa in produzione. I colori dei blocchi si riferiscono a 3 macrofamiglie di applicazione.

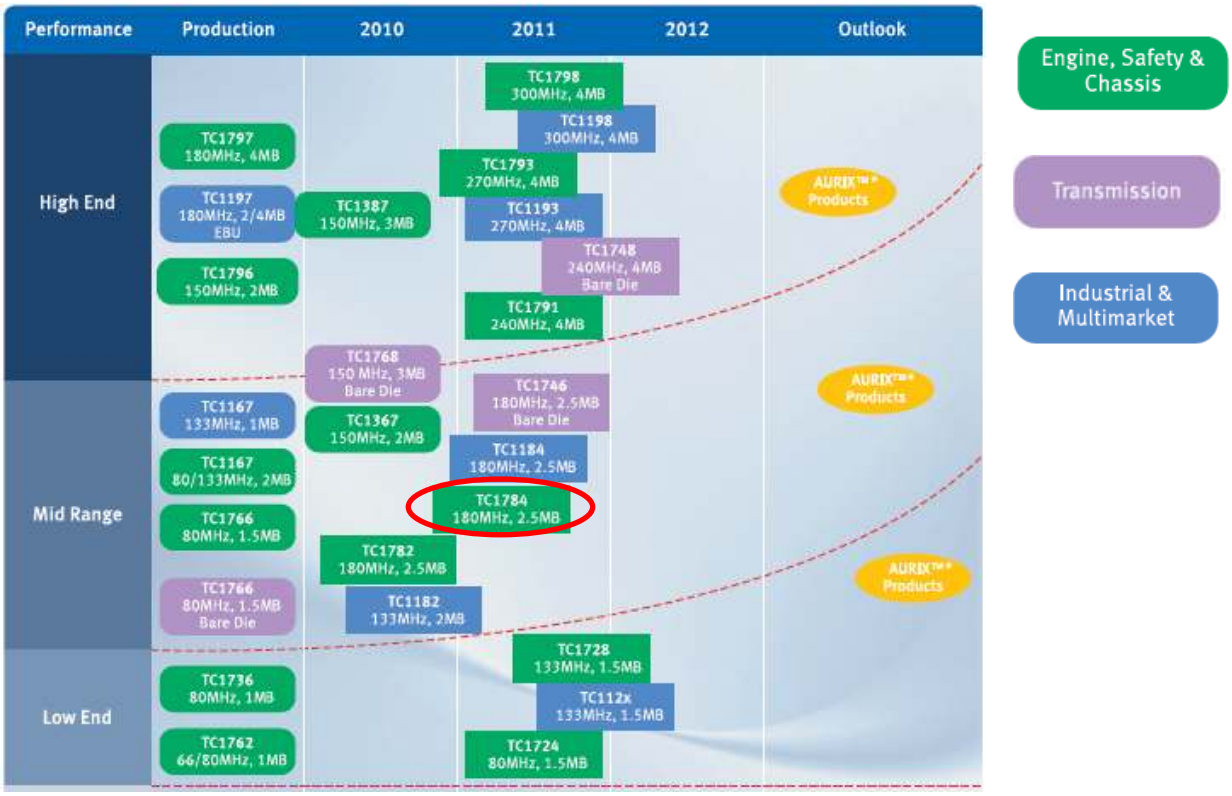


Figura 11: Rappresentazione della famiglia di semiconduttori Infineon a 32 bit

In questa fase ho studiato inoltre il funzionamento generale dell'FTOS in modo da familiarizzare con il prodotto per capirne i meccanismi e gli eventuali punti critici.

3.1.1 Functional Test Operating System (FTOS)

Le caratteristiche dell'FTOS possono così essere riassunte:

- supportare l'esecuzione delle TAs;
- supportare la comunicazione (utilizzando un protocollo a livello alto per permettere un grande trasferimento di dati);
- fare da collettore e rendere disponibili i risultati dei test;
- capire le differenze hardware tra i diversi DUT;
- supportare il debug delle TAs in tutte le modalità(step by step, skip, breakpoints,..);
- gestire eventi straordinari;
- supportare l'allocazione di risorse;
- supportare l'esecuzione anche in assenza di appropriate risorse DUT.

E' possibile inoltre fare un confronto tra i diversi sistemi usati nel TestWare e i protocolli del modello ISO/OSI, una descrizione astratta dei livelli di comunicazione nei disegni dei protocolli di rete, sviluppato come una parte dell'iniziativa dell'Open Systems Interconnections (OSI). Nella sua forma più basica, il modello suddivide l'architettura di rete in sette livelli che, elencati dall'alto verso il basso sono: applicazione, presentazione, sessione, trasporto, rete, collegamento dati e strato fisico. Lo scopo di ciascun livello è fornire dei servizi ai livelli superiori, mascherando come ciò avvenga. Ogni livello passa delle informazioni e dei dati ai livelli sottostanti, fino a quando viene raggiunto il livello fisico che trasmette.

Figura 12: Rappresentazione schematica del modello ISO-OSI

Le funzioni dei singoli livelli si possono riassumere come:

1. Il livello fisico è lo strato più basso del modello ISO-OSI e coordina le funzioni richieste per trasmettere un flusso di bit in un canale di comunicazione.
2. Il livello dati è il responsabile della trasmissione di pacchetti di dati tra nodi adiacenti. La trasmissione deve essere priva di errori non segnalati. Questo livello deve identificare l'inizio e la fine dei pacchetti, gestire le eventuali ritrasmissioni e il controllo di flusso.
3. Lo strato rete è responsabile dell'instradamento dei pacchetti dalla sorgente alla destinazione attraverso i vari nodi del sistema. Questo livello deve conoscere la topologia della rete e scegliere il cammino migliore per fare arrivare il messaggio a destinazione.
4. Il livello quattro è lo strato di trasporto ed è il responsabile per la spedizione di un messaggio da un sistema ad un altro. Questo livello dà la destinazione di ogni pacchetto di dati considerando il singolo pacchetto in modo individuale. Il livello di rete assicura che il messaggio arrivi intatto e con il controllo del processo e degli errori. Il livello trasporto deve inoltre gestire connessioni multiple all'interno dello stesso elaboratore.

5. Il livello sessione è il livello organizzatore della rete di dialogo: stabilisce e sincronizza l'interazione tra i sistemi di comunicazione e il conseguente scambio di dati.
6. Al livello presentazione è affidata la gestione della sintassi dell'informazione da trasferire.
7. Il livello applicazione è responsabile per l'accesso alla rete da parte degli utenti. Questo livello provvede a interfacciare l'utente e fornisce altri servizi di supporto.

In questo modello FTOS copre i livelli centrali: esso fa da collettore per i messaggi di comando trasmessi dagli ATE e spedisce questi dati alle TAs, inoltre gestisce le sessioni di test e presenta i risultati con una determinata struttura.

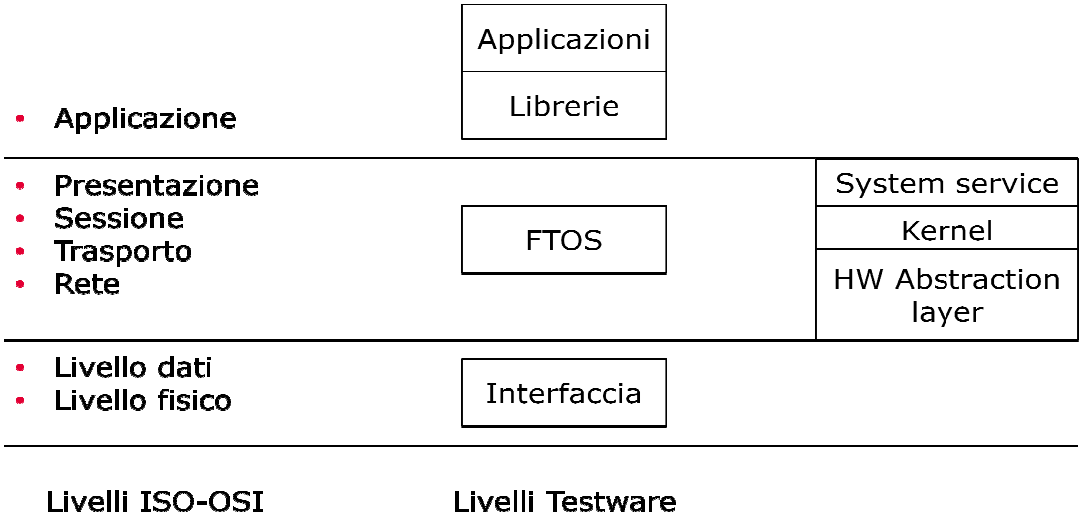


Figura 13: Confronto TestWare con il modello ISO-OSI

Tutte queste funzionalità sono implementate usando tre strati TestWare:

- System Services layer
- Kernel layer
- HW abstraction layer

Il livello System Services esegue tutte le operazioni dell'FTOS. Questo livello può eseguire i comandi con un "motore" che effettua i test sulla Flash e può gestire le trappole hardware e le chiamate a sistema delle TAs. Tutti questi requisiti sono all'interno del sistema System Services e sono inseriti qui dai livelli più bassi dell'FTOS.

Il Kernel è lo strato centrale dell'FTOS e lo si può considerare come un ponte tra l'interfaccia dell'applicazione e i dati processati dal livello hardware. Tra le responsabilità del Kernel ci sono il controllo del sistema di dati e la gestione della sessione in tutte le sue fasi, dall'inizializzazione al caricamento dei risultati passando per l'esecuzione.

Lo strato hardware astratto nasconde i dettagli dell'implementazione di particolari settaggi hardware e delle funzionalità di comunicazione, generalizzando i modelli e gli algoritmi. Queste generalizzazioni derivano da similitudini tra i dispositivi che possono essere inglobate in un modello.

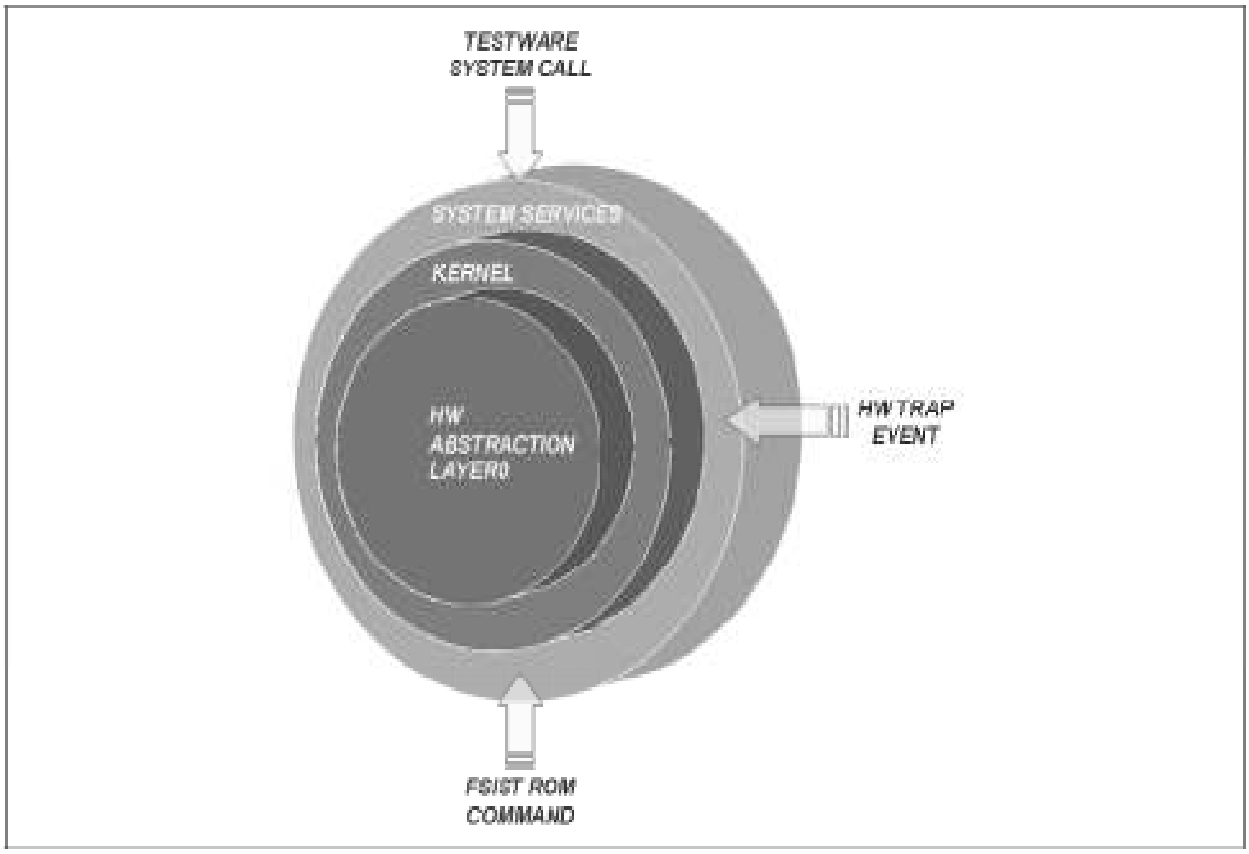


Figura 14: Diagramma rappresentativo dei livelli dell'FTOS

3.2 Studio di fattibilità

Nella seconda fase ho studiato più dettagliatamente le funzionalità da testare.
Prima di ipotizzare un flusso di regression per la singola funzionalità è stato necessario studiarne il meccanismo in ogni dettaglio. Dopo aver capito il funzionamento, ho creato una presentazione in Power Point, detta *Implementation spec* in cui ho illustrato i punti salienti della caratteristica da testare ed ho proposto un flusso di regression. A questo punto la mia ipotesi di regression è stata analizzata e discussa con gli sviluppatori FTOS e con loro abbiamo valutato il *test coverage* e l'eventuale aggiunta di alcuni casi particolari da approfondire.

3.2.1 Comando Set PLL

La prima regression di cui ho studiato la fattibilità è stata sul comando di set del *Phase Locked Loop (PLL)*.
Questo comando fa parte dei comandi *Operating System(OS)*, comandi supportati dai moduli a basso livello del Kernel. Il livello basso Kernel copre gli strati di Rete e di Trasporto del modello ISO-OSI con alcune tabelle strutturate che contengono la sequenza e i dati utilizzati durante l'esecuzione di tutti i test.
I comandi dell'OS consentono di configurare l'FTOS e controllare lo stato dei DUT oltre che supportare l'esecuzione dei test dopo la fase di inizializzazione.
I comandi OS sono strutturati in 32 bit: nei primi bit è presente il comando in sé, mentre in quelli successivi è inserito il valore del sottocomando e di eventuali parametri.
Il comando di set del PLL scrive i valori corretti dei divisori del PLL nei registri di competenza. Dopo aver studiato il funzionamento del comando di set del PLL ho creato un flusso di regression che effettuasse tutti i necessari controlli per evidenziare il corretto funzionamento del comando.
Ad esempio, sapendo che nel momento in cui viene eseguito un comando il PLL deve essere "lock", la regression deve includere un controllo per vedere se il *Voltage Controlled Oscillator (VCO)* è "lock". Questa condizione è stabilita da un flag in un registro specifico che è settato (=’1’).
Il PLL può funzionare in 4 modalità differenti:

- Normal mode: la frequenza in ingresso è divisa per un fattore P, moltiplicata per un fattore N e divisa nuovamente per un fattore K
- Bypass mode: la frequenza in ingresso al PLL è connessa direttamente all'uscita.
- Prescaler mode: la frequenza in ingresso è divisa per un fattore K
- Freerunning mode: la frequenza in uscita dal VCO è divisa per un fattore K

Quando il PLL è utilizzato dal l'FTOS deve essere in normal mode e quindi un possibile controllo da fare è sui valori di N, P e K.

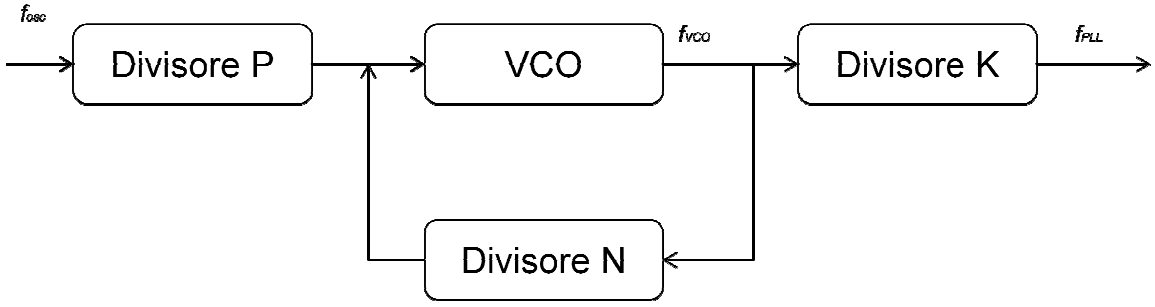


Figura 15: Schema a blocchi di un PLL in normal mode.

Nel comando di set del PLL si va ad impostare un valore della frequenza e il DUT va a prelevare da un file di testo i valori da inserire come N, P e K nei registri dedicati al PLL.
A questo punto si può creare un possibile flusso di regression:

1. Download FTOS nel dispositivo;
2. Impostare la frequenza desiderata;
3. Eseguire il comando di set PLL;
4. Controllare i valori N, P, K;
5. Controllare se il PLL è "lock";
6. Ripristino delle condizioni iniziali.

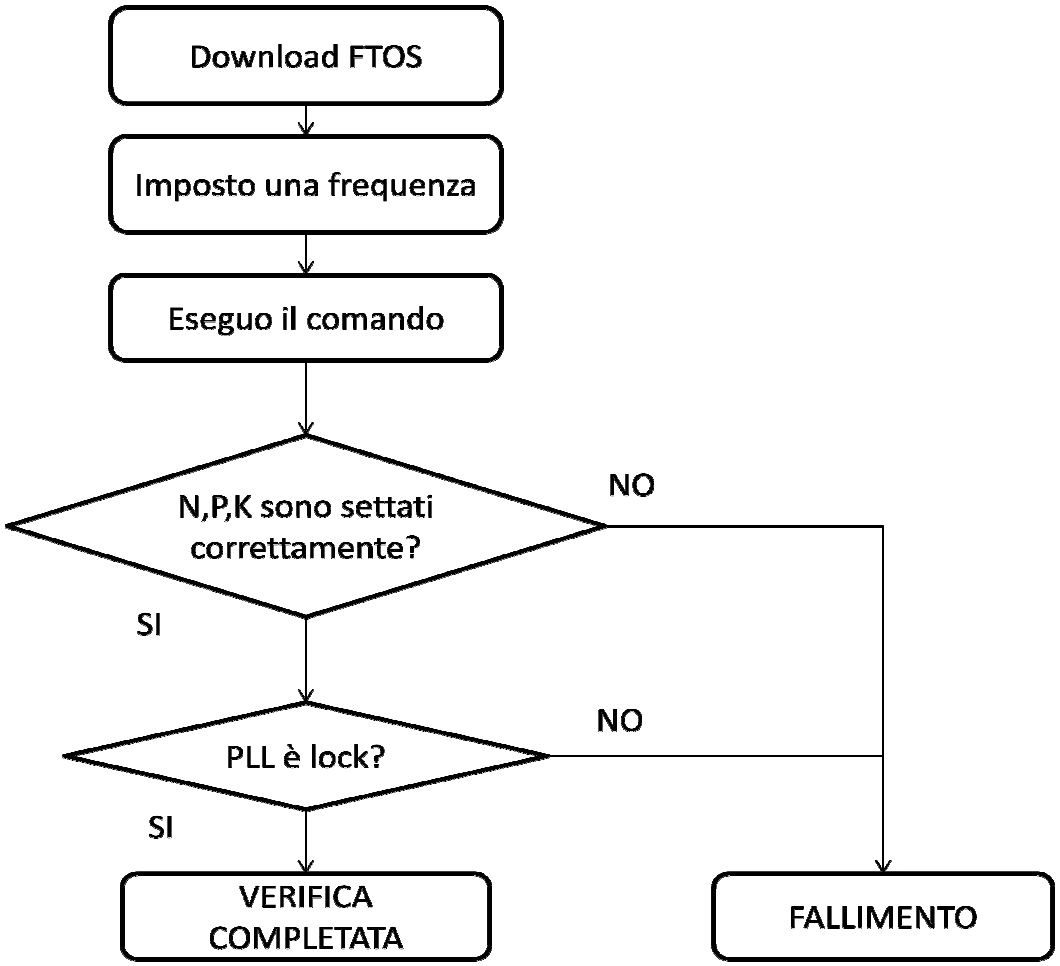


Figura 16: Diagramma di flusso del test di Regression sul PLL

Tutto questo viene inserito nell'Implementation Spec in modo che sia chiaro come e perché sia stata creata la regression e cosa ci si prefigge di controllare.

3.2.2 Il comando Check CRC

Il controllo a ridondanza ciclica è un metodo molto utilizzato per trovare errori di codice ed è comunemente utilizzato nelle reti di comunicazione. Il *Cyclic Redundancy Check (CRC)* genera una stringa di bit di controllo che viene trasmessa assieme ai dati, i quali vengono elaborati attraverso una rete logica. Il CRC, in questo caso, viene calcolato sulla test application. Esistono diversi tipi di CRC e ciò che li differenzia l'uno dall'altro è il grado del polinomio generatore: i CRC più utilizzati sono il CRC8, CRC16 e CRC32.

Il comando Check CRC è anch'esso un comando dell'Operating System e viene utilizzato per controllare il corretto funzionamento del CRC.
Questo comando utilizza i valori inseriti in alcuni registri della memoria RAM, quindi un possibile test di regression si baserà sulla modifica di questi valori e sulla verifica del funzionamento del comando.
Il metodo per il calcolo del CRC utilizza l'indirizzo di partenza della TA e la sua dimensione e, eseguendo il comando, viene calcolato un valore del CRC e confrontato con il risultato aspettato.
Non sapendo in principio il valore esatto del CRC, ma essendo a conoscenza del fatto che quando il comando fallisce inserisce in un registro il valore corretto, il test di regression inizia con il fallimento del comando. Per far sì che il comando fallisca è quindi sufficiente inserire un valore errato del CRC nel registro nel quale è presente il numero aspettato. In questo caso, un possibile flusso di regression è:

1. Download FTOS nel dispositivo;
2. Inserimento di un valore errato nel registro del CRC;
3. Esecuzione del comando di Check CRC (il comando deve fallire);
4. Salvataggio del valore corretto;
5. Esecuzione del comando di Check CRC (il comando deve passare);
6. Ripristino delle condizioni iniziali.

Anche in questo caso, tutto questo viene inserito nell'Implementation spec e verificato con i responsabili di FTOS.

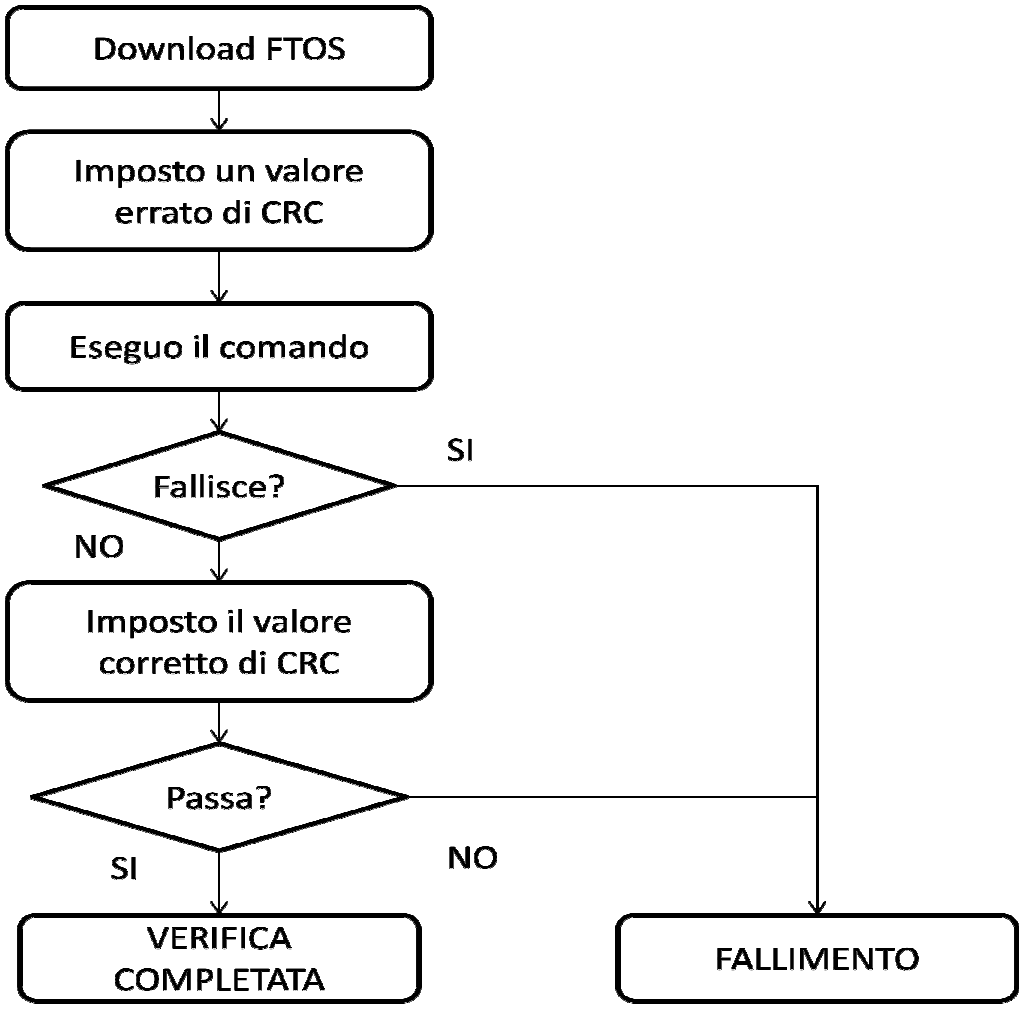


Figura 17: Diagramma di flusso del test di Regression sul comando Check CRC

3.2.3 Il comando Delay

Il comando di Delay, come i precedenti, fa parte dei comandi dell'Operating System.
FTOS permette di salvare il tempo trascorso, durante l'esecuzione del flusso, in un registro dedicato utilizzando il timer di sistema: un contatore legato direttamente alla frequenza di sistema f_{sys}, abilitato al momento dell'accensione del dispositivo. Il sistema utilizza al suo interno sette diversi timer concatenati da 32 bit l'uno, in modo tale da raggiungere tempi registrati molto elevati.
Il valore raggiunto dal contatore non può essere modificato, ma solamente essere letto.
Principalmente con il comando di delay viene eseguita una *non operation (nop)* per un numero di volte che viene impostato come parametro del comando.
Il tempo che il sistema impiega ad eseguire una nop deve essere costante sia che ne venga eseguita solamente una sia che ne vengano eseguite molte. Da questo nasce un'ipotesi di regression: variando il numero di nop da eseguire o la frequenza del sistema,

devo ottenere i corrispettivi multipli del tempo di esecuzione. In questo caso, un possibile flusso di regression è:

1. Download FTOS nel dispositivo;
2. Esecuzione di 500 nop;
3. Salvataggio del tempo di elaborazione;
4. Esecuzione di 1000 nop;
5. Salvataggio del tempo di esecuzione;
6. Diminuzione/Aumento della frequenza di sistema
7. Salvataggio del tempo di esecuzione;
8. Verifica della correttezza dei dati;
9. Ripristino delle condizioni iniziali.

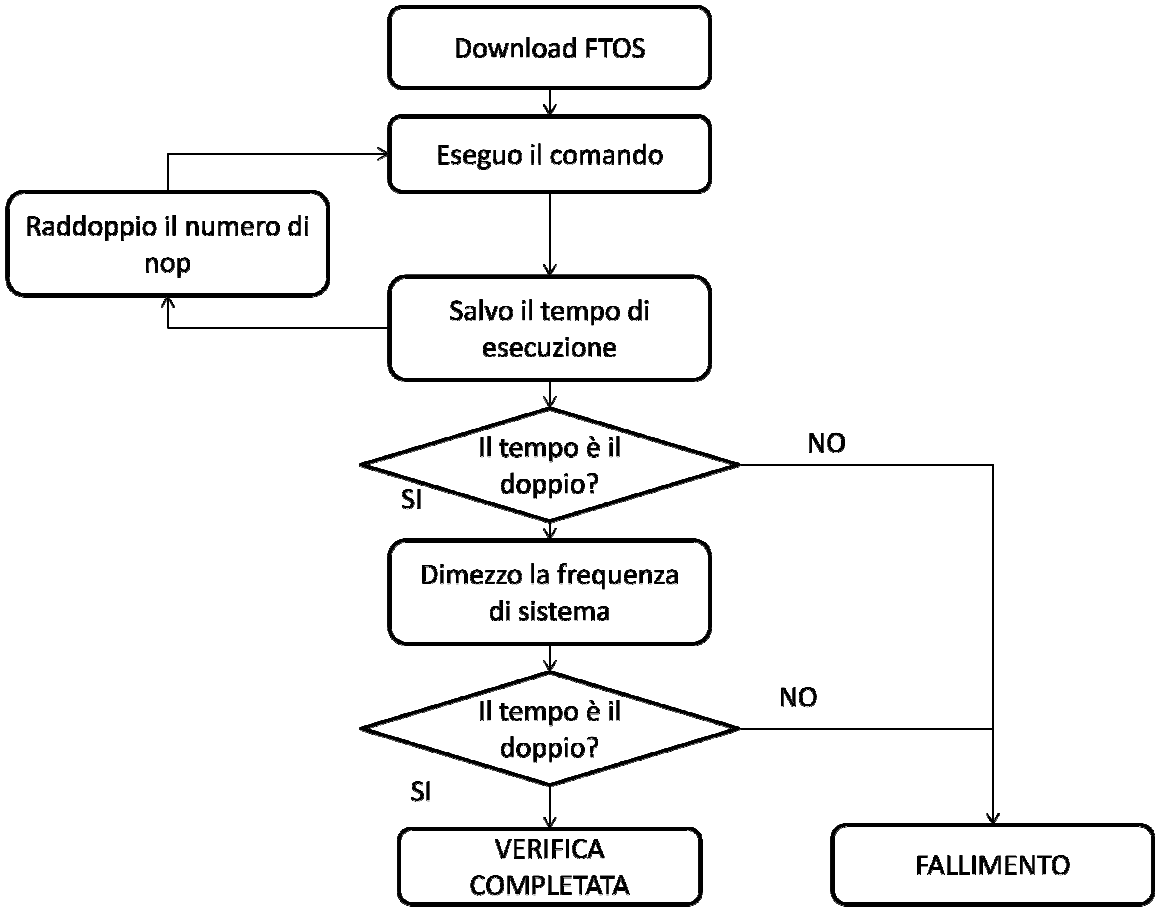


Figura 18: Diagramma di flusso del test di Regression sul comando Delay

Questo studio viene inserito nell'Implementation spec e, come nei casi precedenti, verificato con i responsabili dell'FTOS.

3.3 Scrittura e debug del codice

Terminata la fase di studio e dopo aver creato l'Implementation Spec, inizia la fase di scrittura del codice. In questa fase, ho iniziato a creare i programmi di regression a partire dal comando di set del PLL. Durante la scrittura del codice, sono state necessarie alcune modifiche del flusso di regression: ad esempio nel caso del comando di set del PLL, prima di eseguire il comando era necessario settare i valori della frequenza in tutte le modalità di funzionamento.

Durante questa fase la programmazione va di pari passo con il debug in quanto possono essere fatti sia degli errori di sintassi sia errori logici. Nell'affrontare il debug del codice è risultata importante l'ottima integrazione di Jazz con l'editor di testo Eclipse. In questo modo, nel momento in cui un programma di regression non funzionava correttamente, è stato possibile eseguire il debug passo passo. In Eclipse, ogni volta che si avanza di uno step, è presente una finestra che segnala il valore di tutte le variabili ed evidenzia quando queste cambiano rendendo di facile lettura le modifiche che avvengono nella memoria.

Nella fase di sviluppo del codice è avvenuto difficilmente di ottenere il risultato desiderato senza aver dovuto debuggare e correggere gli errori. Questo perché un errore di sintassi o un errore logico accadono facilmente, soprattutto non conoscendo ancora a fondo il dispositivo.

La regression più complessa che ho incontrato è stata quella riguardante il comando di Check del CRC. La complessità non è stata relativa all'algoritmo da creare, forse il più semplice, ma dovuta al fatto che era presente un baco dell'FTOS: in questo caso quindi la scrittura della regression si è rivelata fondamentale per eliminare il baco.

Questa regression doveva controllare il funzionamento del comando di Check del CRC e, il flusso di download che avevo deciso, prevedeva di eseguire il comando dopo aver impostato un valore errato di CRC atteso. In questo modo il comando doveva fallire e scrivere in un'area di memoria dedicata, chiamata Debug Table, il valore corretto del CRC. Il problema è nato proprio da questo: nell'indirizzo previsto della Debug Table non veniva scritto nulla. A questo punto o era presente un errore nel mio codice che doveva generare il fallimento, oppure era presente un baco in FTOS. Per verificare che effettivamente non avevo fatto un errore nella mia regression è stato possibile analizzare, con l'utilizzo di Jazz, direttamente le locazioni di memoria del dispositivo su cui è anche possibile scrivere. In questo modo ho ricreato la stessa situazione che generavo con la scrittura del codice e mi sono assicurato che il problema era dovuto ad un baco di FTOS.

Ho segnalato il fallimento ai responsabili di FTOS che hanno sistemato il baco: La regression eseguita sul nuovo FTOS funzionava correttamente ed ha dimostrato tutta la sua utilità.

Eseguendo poi la regression sul comando set del PLL ho trovato un errore sulla documentazione, in quanto era presente una discrepanza tra quello che rilevavo

sperimentalmente e quello che invece era scritto sulla documentazione. In questo caso ho segnalato l'errore ai responsabili della documentazione per correggerla.

3.4 Risultati e caricamento in RegGAE©

Dopo aver terminato la scrittura del codice, all'esecuzione del programma viene creato in Jazz un "Datalog" che riassume in modo leggibile i risultati ottenuti e rispecchia il flusso di regression eseguito.
Ogni test di regression ha un Datalog dedicato e solitamente, dopo aver eseguito in sequenza i test richiesti, si invia il Datalog al responsabile del prodotto che si è testato.

```
00000   ****************************************************
00000   ***********_TestCase_FTOS_PLL_***********
00000   ****************************************************
00000
00000   TEST_in_MONITOR:Command_0x30_0x2000_PASS:_0_ms
00000   PASS:_VCO_is_locked_
00000   At_frequency:_80_
00000   N_value_is_AS_EXPECTED:_0x40
00000   P_value_is_AS_EXPECTED:_0x02
00000   K_value_is_AS_EXPECTED:_0x08
00000   TEST_in_MONITOR:Command_0x30_0x2000_PASS:_0_ms
00000   PASS:_VCO_is_locked_
00000   At_frequency:_160_
00000   N_value_is_AS_EXPECTED:_0x30
00000   P_value_is_AS_EXPECTED:_0x02
00000   K_value_is_AS_EXPECTED:_0x03
00000   TEST_in_MONITOR:Command_0x30_0x2000_PASS:_0_ms
00000   PASS:_VCO_is_unlocked_as_expected
00000   At_frequency:_FRM_
00000   N_value_is_AS_EXPECTED:_0x30
00000   P_value_is_AS_EXPECTED:_0x02
00000   K_value_is_AS_EXPECTED:_0x03
00000 P -01
00001   Test_duration_11.921_
```

Figura 19: Esempio di Datalog nel caso del comando di set del PLL

In questo modo si ha un ulteriore feedback sul lavoro eseguito ed eventualmente si modifica il programma per rendere il Datalog più comprensibile oppure può essere necessario aggiungere delle casistiche da analizzare per rendere il test più completo. Ad esempio, nel caso della regression sul comando di Check del CRC è stata richiesta una modifica in quanto il Datalog comprendeva solo i risultati ottenuti, mentre non era comprensibile cosa veniva fatto realmente.

```
00000   ****************************************************
00000   ***********_TestCase_FTOS_CRC_***********
00000   ****************************************************
00000
00000   PASS:FAIL_as_Expected:Command_0x30_0xC000_done:_0_ms
00000   PASS:PASS_as_Expected:Command_0x30_0xC000_done:_0_ms
00000 P -01
00001   Test_duration_8.969_
```

Figura 20: Datalog del comando di Check del CRC di cui è stato richiesto un miglioramento

Il codice TestWare è posto sotto controllo di versione tramite un software di versionamento. In questo modo si mantiene traccia di tutte le modifiche apportate che, fra l'altro, sono disponibili in ogni momento a tutto il team di sviluppo. Nel caso venga inserita una modifica "dannosa" è sempre possibile tornare alla versione precedente del codice.

A questo punto, con le tre regression funzionanti e i risultati sistemati, si possono inserire i test realizzati nel sistema automatico di regressione (RegGAE©) vero scopo dei test di regression.

Al termine della giornata lavorativa il team del TestWare avrà eseguito un certo numero di modifiche al codice. Per testare che le modifiche non abbiano introdotto bachi o malfunzionamenti, ogni notte viene eseguita una compilazione del codice rispettiva per ogni microcontrollore. Come output di questa sessione si avranno tanti file di FTOS quanti dispositivi sono supportati. A questo punto RegGAE© si occupa di scaricare i compilati in ogni dispositivo e di far eseguire tutti i test di regression presenti nella libreria dedicata.

Con questo flusso di lavoro, le regression vengono eseguite tutte le notti con un riscontro diretto sull'hardware (microcontrollore) e nel caso una o più di esse dovesse fallire, si avvertirebbe il responsabile del prodotto del malfunzionamento in modo da prendere le misure correttive.

4 CONCLUSIONI

Considerando che il testing dei dispositivi viene fatto attraverso il TestWare e che i test di regression vanno a stabilire la loro correttezza, si può facilmente comprendere come l'aver inserito i test di regression nel ciclo produttivo di Infineon si è rivelato molto utile principalmente per:

- la netta riduzione del tempo che intercorre fra l'inserimento di un baco e la sua scoperta e correzione,
- il considerevole aumento della qualità del software rilasciato ai siti di produzione dei dispositivi,
- la robustezza che ora accompagna i dispositivi testati.

I test di regression sono molto utili e danno un verifica di sistema molto più ampia rispetto allo *unit testing* che va a verificare una singola funzionalità in modo separato. Una singola funzionalità può, ad esempio, essere eseguita correttamente ma avere all'interno del codice un errore che va a modificare un'area di memoria che non è di sua competenza. Con lo unit test questo errore non può essere trovato, mentre viene individuato con i test di regression avendo un più ampio raggio di azione.

Nel caso specifico, le regression per FTOS sono estremamente necessarie per accrescere notevolmente l'affidabilità del sistema. In questo modo, essendo le TAs gestite e supportate da FTOS, nel momento in cui si dovesse verificare un fallimento, si concentrano le proprie forze su un campo d'azione più ristretto, poiché si suppone che a fallire, con questo flusso di lavoro, non sia FTOS sebbene la certezza non si avrà mai.

Nel mio caso l'efficacia delle regression sì è poi dimostrata sul campo visto che i miei script hanno portato all'individuazione di due bachi su tre test realizzati: uno nel sistema stesso ed uno nella documentazione.

Le funzionalità da testare rimangono molte e solo realizzando un test specifico per ognuna di esse, diminuirebbe drasticamente la probabilità di un errore di FTOS. Oltre a questo, riuscendo a coprire il maggior numero di funzionalità possibile, le regression verrebbero utilizzate per il loro scopo principale e cioè essere inserite nel tool automatico che le esegue ogni notte. In un flusso di lavoro così costruito, nel momento in cui viene modificato FTOS, si ha un riscontro pressoché immediato di eventuali bachi, senza così rischiare di generare codice costruito sopra un errore che alla lunga sarebbe di difficile individuazione.

Lavorare nel team MC di Infineon di Padova è stata un'esperienza decisamente stimolante, con un'altissima competenza da parte di tutti i componenti del gruppo sempre disponibili ad aiutarmi.

5 BIBLIOGRAFIA

[1] Francesco Bonacina. Regression Suite Tutorial, Documento interno Infineon, Dicembre 2011.

[2]Antonio Fin, Massimo Atti, Paolo Raimondi, Alessandro Scanferla. FTOS User Manual, Documento interno Infineon, Aprile 2010.

[3]Antonio Fin . TestWare Documentation, Documento interno Infineon, Gennaio 2011.

[4]Deborah Lunardon, Alessandro Scanferla. Software and Hardware tools for embedded Flash Memories characterization 32-bit embedded Flash Memories, Documento interno Infineon, Febbraio 2011.

[5]Antonio Fin. FTOS Overview, Documento interno Infineon, Aprile 2010.

[6] Internal Target Specification of TC1784-AudoMax, Documento Interno Infineon.

[7] Sito web principale del Perl. www.Perl.com

[8]Cay S Horstmann. Concetti di informatica e fondamenti di java, Terza edizione, Apogeo education, giugno 2005.

[9] E. Clayberg, D. Rubel. 'Eclipse Plug-ins', 3rd Ed., Addison-Wesley 2008.