Implementació i simulació funcional en MATLAB® del protocol IEEE 802.15.4 de WSN

Memòria del projecte
d'Enginyeria en Informàtica
realitzat per
Miquel Izquierdo Ustrell
i dirigit per
Jordi Carrabina i Eloi Ramón
Bellaterra, 16 de Juny de 2008

El sotasignat, Jordi Carrabina Bordoll
Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Miquel Izquierdo Ustrell

I per tal que consti firma la present.

Signat: .....
Bellaterra, 15 de Juny de 2008

ÍNDEX

Capítol 1. Introducció

En aquest apartat es posa en context el treball realitzat explicant les idees generals, la situació actual, l'objectiu que es pretén assolir i l'estructura d'aquest document.

1.1 Conceptes

Abans d'entrar a estudiar el desenvolupament que s'ha dut a terme en aquest projecte convé incloure algunes idees bàsiques. I per això, a continuació, s'analitzen els conceptes del seu entorn tecnològic i, breument, com es relacionen amb la tasca realitzada.

1.1.1 Simulació

La idea de simulació està íntimament lligada amb els objectius d'aquest projecte, però que s'entén per simulació?. L'entrada corresponent a l'enciclopèdia catalana diu "*Representació d'un fenomen de naturalesa física, tècnica, biològica, psicològica, militar o econòmica mitjançant un model físic o matemàtic que en permet una anàlisi més senzilla, econòmica o innòcua que si aquesta es realitzés sobre l'original...*"[1]. Per tant, el propòsit d'aquest treball ha de ser imitar un conjunt de fenòmens, en aquest cas els propis de les xarxes de sensors. A més, ja es pot preveure que entre els resultats que es pretenen obtenir s'hi troba de forma implícita la intenció de simplificar i fer més econòmic l'anàlisi d'aquests fenòmens.

Per "entorn de simulació" s'entén el software encarregat del processament i la interacció que requereix el model de simulació. En aquest cas convé aclarir que, a diferència de molts altres softwares de simulació disponibles al mercat, el que es pretén desenvolupar és un model que estigui lligat a un entorn ja existent, no específic de les comunicacions.

En aquest cas l'entorn triat és MATLAB® , per a ser més exactes Simulink® , eina inclosa en el primer. Els motius de l'elecció es detallen més endavant (veure apartat 1.3.2).

MathWorks, companyia desenvolupadora de MATLAB® va ser fundada l'any 1984 i des d'aleshores ha anat ampliant el conjunt d'eines d'aquest programari [2]. Més recentment han inclòs utilitats per a treballar amb diagrames d'estats i per a treballar conjuntament amb el llenguatge de programació 'C', en les quals s'ha centrat aquest projecte.

Per a ser precisos, cal concretar que treballarem amb autòmats finits els quals es defineixen com a "*Model matemàtic d'un sistema que té un nombre finit d'estats d'entrada i de sortida... i en el qual l'estat de sortida depèn en qualsevol moment de l'entrada present i dels estats interns...*"[3]. Aplicant-t'ho als requisits que tenim, i mirant-t'ho de forma simplificada, tenim unes dades a ser transmeses, les entrades, que es processen depenent dels estats interns, els paràmetres de la xarxa, i s'entreguen preparades per a ser enviades, les sortides.

1.1.2 Comunicació

La comunicació és un acte que s'utilitza constantment i en molts àmbits. En un context general es pot explicar com a "*Transmissió d'informació d'un punt a un altre mitjançant un enllaç telefònic, telegràfic, radioelèctric, etc... consta de cinc apartats: I) Una font d'informació... II) El transmissor... III) El canal... IV) El receptor... V) La destinació*". Per tant es pot extreure d'aquesta que s'ha de reproduir dins de l'entorn de simulació cadascun dels anomenats apartats. En l'anterior definició es troben a faltar alguns detalls, sobretot que cada etapa d'aquesta transmissió de dades es realitza seguint un conjunt de regles que governen

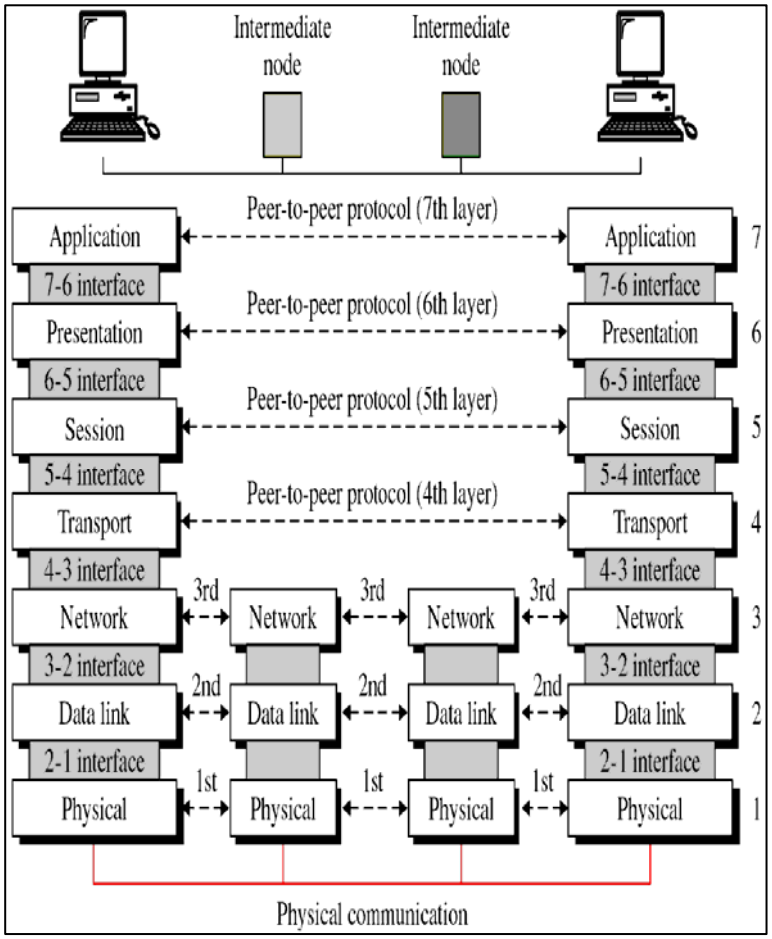


Figura 1.1 Diagrama de capes estàndard OSI

l'intercanvi de dades entre els dos punts [4], les quals formen el que coneixem com a protocol.

Per a reduir la complexitat del disseny, la majoria de xarxes s'organitzen com un conjunt de capes o nivells. El nombre de capes, el nom i la funció poden varia a cada xarxa, el model de capes de referència per al desenvolupament de nous estàndards és l'OSI [5], veure figura 1.1. A cada nivell s'utilitza un protocol, per tant en una comunicació intervenen un conjunt de capes i protocols, això es coneix com a arquitectura de xarxes [6].

Existeix una gran quantitat d'arquitectures diferents, en aquest cas es treballa seguint l'estàndard IEEE 802.15.4 que és una de les arquitectures més utilitzades en les xarxes de sensors sense fils.

Degut a la gran quantitat de feina que representa cadascun d'aquests nivells, s'ha seleccionat el conegut com a Capa d'Accés al Medi (MAC). Aquest s'encarrega de les funcions bàsiques que es descriuen en el següent apartat.

1.1.3 IEEE 802.15.4

Aquesta arquitectura per a comunicacions sense fils està constituïda per dos nivells el físic i el d'accés, els quals donen serveis a las capes superiors.

Els sistemes de comunicació formats per aquest protocol tenen 2 tipus de dispositius els "full-function device" i els "reduced-function device", més simples. En funció del seu tipus, els dispositius poden realitzar una o varies de les següents tasques a dins de la xarxa: "***PAN coordinator***", "***coordinator***" o "***end device***".

En tots els casos hi ha d'haver com a mínim un "***PAN coordinator***", que es diferència dels altres en què és el coordinador encarregat d'iniciar la xarxa i permetre les associacions a

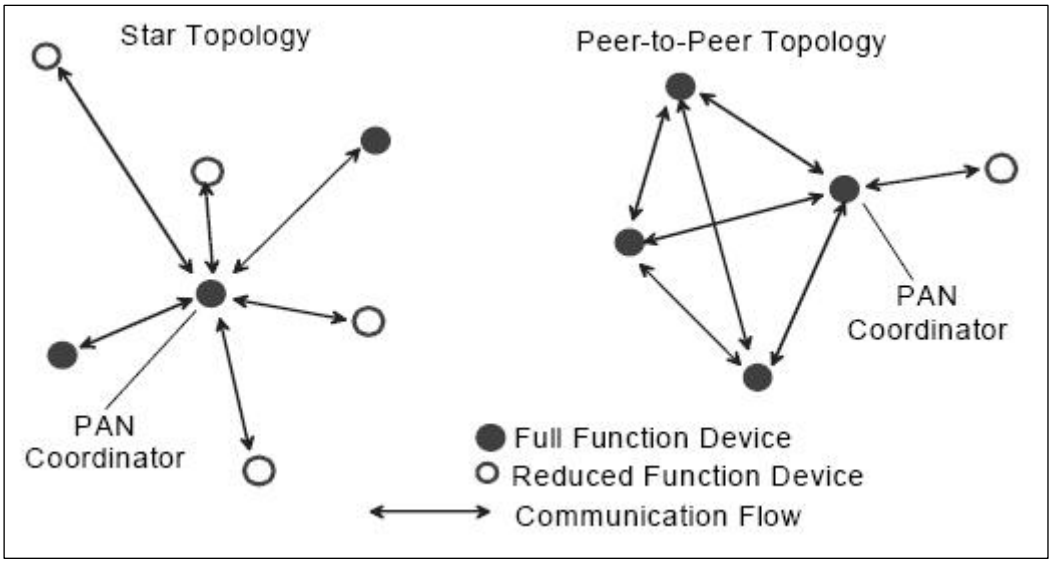


Figura 1.2 Exemples de topologia estrella i punt a punt

aquesta, a més hi poden haver-hi altres "***coordinator***" i "***end device***" creant una topologia en estrella o bé punt a punt, com es pot veure a la figura 1.2.

Les principals diferències entre els dispositius "***coordinators***" i els "***end device***" són que el primer es pot comunicar amb qualsevol dispositiu al seu abast, mentre que el segon només pot intercanviar dades amb coordinadors, això és degut a que els "***end device***" estan pensats per a estar aturats la major part del temps, poden ser alimentats per bateries, mentre que els coordinadors han d'estar sempre actius.

Per a aquest treball s'ha desenvolupat la capa MAC, la qual ofereix dos serveis el "MAC data service" i el "MAC management service". El primer permet l'enviament i recepció de la unitat de dades del protocol MAC, MPDU, a través del servei de dades de la capa inferior, la física. El segon servei fa d'interfície a la MLME-SAP, MAC Level Management Entity – Service Access Point. És a dir, gestiona la transferència de comandes entre el nivell superior i el propi subnivell, el MLME, encarregat de la gestió de la xarxa en aquesta capa. En aquesta gestió s'inclou, entre altres, generació i procés dels *beacons* (paquets de sincronització i configuració), accés al canal, associació i dissociació de la xarxa, confirmació de recepció de paquets (*acknowledges*), etc [7].

En els capítols posteriors s'anirà ampliant aquesta informació a mesura que es faci referència a ella.

1.2 Estat de l'art

Dins d'aquest apartat s'inclouen tres subapartats, el primer tracta l'entorn en que s'utilitzen el protocol 802.15.4 i els simuladors d'aquest. El segon es centra en els simuladors actuals coneguts i les seves principals deficiències. Per últim es detallen les alternatives disponibles en referència als recursos utilitzats.

1.2.1 Context

L'estàndard IEEE802.15.4 existeix des de l'any 2003, data en que es van crear dos grups de treball, 802.15.4 a i b, el primer es va dedicar a la caracterització d'una capa física alternativa, mentre que el segon es va encarregar de les millores que van donar lloc a la versió de l'any 2006 [9].

Aquest estàndard és molt utilitzat, tan per si sol com en conjunció amb altres que descriuen les capes superiors, algunes d'aquestes capes també estan estandarditzades, com per exemple Zigbee, i altres no com MiWi.

La connexió amb Zigbee i la situació en aquesta del 802.15.4 es pot veure a la figura 1.3. Aquests s'utilitzen sobretot en entorns no molt extensos, tot i que afegint les capes superiors això pot variar, per monitoritzar la informació donada per petits sensors, un exemple en el qual s'està implementant és el de companyies com Control4 o AMX que l'apliquen a les cases intel·ligents [8].

Els simuladors de protocols són una eina utilitzada principalment en la modelització de xarxes abans de ser implantades o bé en entorns d'investigació. Actualment n'existeixen tant versions de pagament, com OPNET Modeler®, com de basades en codi obert, com OMNeT++, GloMoSim desenvolupat per la Universitat de Califòrnia, o el més conegut el ns2. Molts d'aquests són simuladors de successos discrets, i ofereixen la possibilitat de reproduir el comportament de diferents models ja inclosos o de desenvolupar-ne de nous, seguint les directrius marcades per cadascun.

A part dels anteriors comentats també n'existeixen molts altres, tots amb unes característiques generals similars, encara que estiguin desenvolupats amb diferents llenguatges i segueixin mètodes diferents per a la simulació.

1.2.2 Mancances

Cadascun dels simuladors comentats en l'apartat anterior tenen les seves pròpies mancances que no s'entraran a discutir, sinó que s'estudiarà la problemàtica general d'aquests.

Aquestes limitacions es poden dividir segons a la part a que afecten, clarament es pot distingir entre les pròpies de capacitat d'emulació i les d'eficiència en l'aspecte de feina que

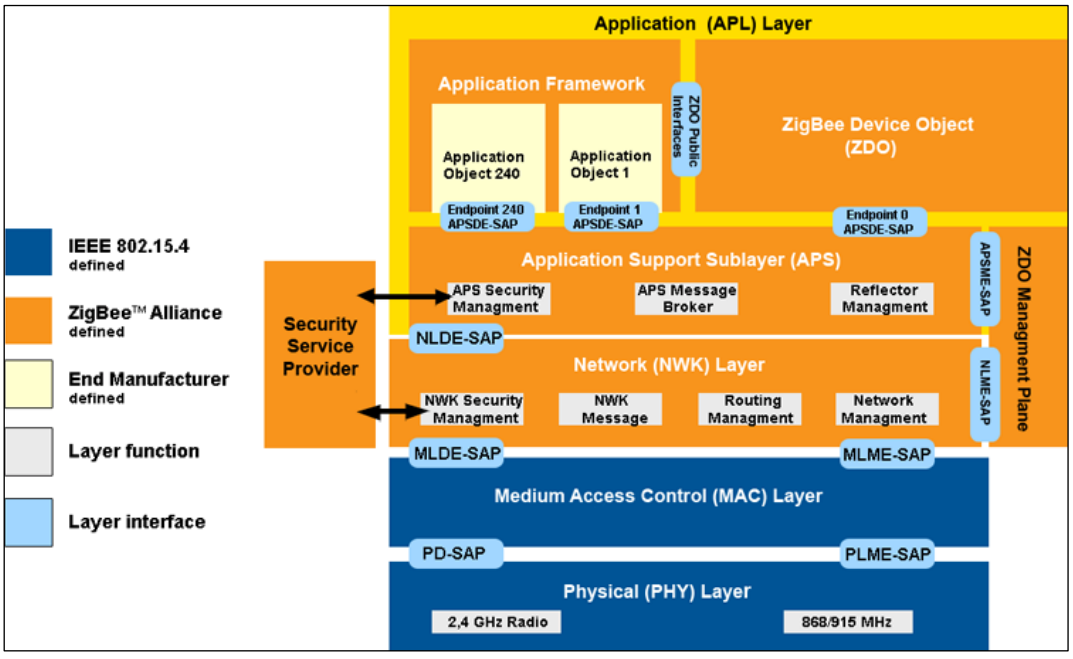


Figura 1.3 Detall de les capes de l'estàndard Zigbee

implica simular un projecte.

En el primer grup es pot ressaltar la falta de la part hardware dins dels possibles models, és a dir, es pot imitar el funcionament del nucli de processament, però en canvi els perifèrics, com "*timers*" o inclús memòries externes, que caldria utilitzar no poden ser reproduïts, això implica diferències que poden arribar a ser significatives, per exemple el temps de lectura i escriptura a les memòries. Aquest tipus d'error s'acumula cada vegada que es produeix un accés, provocant que una petita variació acabi essent important. Un altre cas són les diferències en la precisió d'elements com els *timers* que, pel mateix motiu anterior, pot arribar a ser remarcable.

La que es considera la principal deficiència s'inclou en el segon grup comentat anteriorment, el temps que s'ha de dedicar per a les simulacions, és a dir per a verificar el correcte funcionament del sistema. En un projecte, independentment de la seva dimensió, es requereix un cert nombre d'hores per a generar la simulació, aquestes es multipliquen quan el model no està prèviament generat. Això pot arribar a resultar en el mateix nombre d'hores per a la implementació del model com per a la del dispositiu o sistema real. Aquest problema és molt rellevant sobretot quan el "time to market" és un factor important. El problema està causat per l'obligació d'utilitzar un llenguatge diferent per a les dues situacions, model i disseny, és a dir, que en la majoria de casos no permet aprofitar el codi del simulador per al projecte pròpiament.

Dins d'aquest apartat, també cal tenir en compte que l'ús de dos programes, un per simular i un per implementar, implica si més no estar sotmès a dues llicències. Tot i que el programari lliure és gratuït les llicències poden afegir restriccions a l'hora de comercialitzar un producte.

1.2.3 Alternatives als principals recursos utilitzats

Eines que, entre altres, considerem principals n'he utilitzat dues, la plataforma virtual pròpiament dita, MATLAB® i una implementació de l'estàndard IEEE 802.15.4.

MATLAB® és un entorn amb llenguatge de programació propi orientat a l'anàlisi de dades (i fluxos de dades) i a la computació numèrica (típicament matricial), però el que s'ha utilitzat principalment és el Simulink®, que forma part de l'anterior i és un entorn per a simulació i disseny de models [10]. Dins les alternatives amb unes característiques semblants en tenim de pagament, com Mathematica® [11], i altres de gratuïts, un dels més coneguts és el Octave. També es poden trobar en forma de paquets, com el NumPy o el SciPy per a Phyton. Alguns d'aquests tenen l'inconvenient de no incloure un equivalent a Simulink®. Alguns dels què en disposen són SciLab [12], amb el nom de Scicos, i Modelica.

Les implementacions de l'estàndard IEEE 802.15.4 són molt nombroses, però la majoria són propietàries, gratuïtes o no, d'una companyia lligades a un o varis productes, alguns exemples són TIMAC de Texas Instruments o SMAC de Freescale. A la figura 1.4 es pot veure una comparació de les capes que s'utilitzen en aquest últim amb relació amb 802.15.4 i Zigbee. Per altra banda, també es poden trobar implementacions independents de companyies, normalment en grups de recerca o tesis, com la tesi "Implementation of IEEE 802.15.4 protocol stack for Linux" de Sandeep Sirpatil, de la universitat de North California, o el Open-ZB desenvolupat per el grup IPP-Hurray [13].

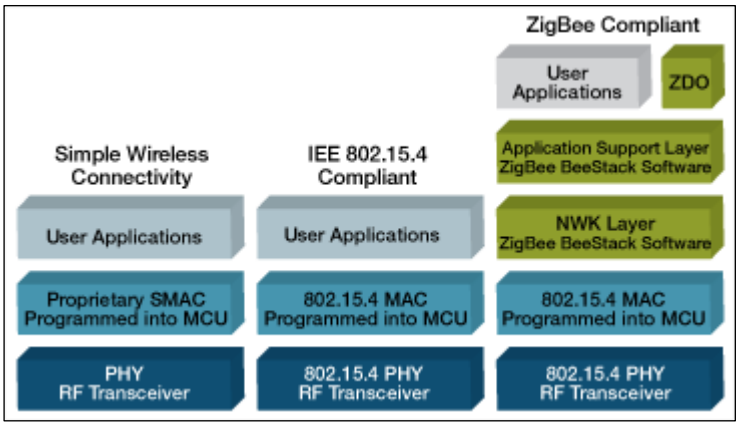


Figura 1.4 Comparació de les capes que s'utilitzen amb SMAC, IEEE802.15.4 i Zigbee

Els motius pels quals s'han triat aquests recursos, entre els considerats viables, es detallen més endavant a l' apartat 1.3.2.

1.3 Objectius i decisions

Aquest apartat es divideix en dos parts, en una primera es detallen els punts que es pretenen obtenir, no només en aquest treball, sinó també alguns propis de la recerca en que aquest s'engloba. Aquests es consideren necessaris per a comprendre algunes de les decisions preses, el conjunt de les quals es detallen al segon apartat.

1.3.1 Objectius

El principal objectiu que es pretén assolir és dissenyar un simulador de la capa MAC, definida a l'estàndard IEEE 802.15.4, que reuneixi les característiques que es detallen en el següent paràgraf. Convé matisar que l'objectiu no és implementar un simulador, sinó utilitzar una plataforma genèrica existent, MATLAB®, i definir sobre ella, una metodologia que, permeti utilitzar-la més com a plataforma específica de desenvolupament i simulació de protocols per a xarxes de sensors, més que no pas com a simple simulador.

Els objectius que es plantegen assolir en el marc d'aquest treball i de la recerca en que s'engloba són els següents:

- Implementació de la capa MAC de l'estàndard IEEE 802.15.4. L'objectiu no consisteix en crear un software més, sinó en aprofitar-ne algun d'existent, i

reconvertir-lo per poder treballar-hi amb ell des de l'entorn Simulink®. Dels existents no se'n coneix cap que compleixi aquests requisits.

- Establir la viabilitat de MATLAB® com a eina de desenvolupament de la plataforma. Aquest és un objectiu simbòlic ja que està suficientment provat que MATLAB®, amb les seves eines, permet dissenyar i crear les implementacions corresponents per a diversos dispositius.
- Permetre accelerar el procés de desenvolupament unificant el codi simulat i el firmware dels dispositius. La majoria de simuladors existents tenen el seu propi llenguatge o estructura per a implementar els dispositius a emular. La intenció és canviar això, és a dir, oferir la possibilitat d'aprofitar al màxim la implementació de la simulació per al firmware dels dispositius. Això implica la utilització d'un llenguatge de programació per al qual la majoria de plataformes disposin de compilador. Per altra banda, un dels avantatges que es pretén aprofitar de Simulink® és la seva capacitat de generar, a partir del model, la simulació i el codi equivalent en llenguatge C. Aquesta propietat la donen les *tool-boxes* Real-Time Workshop® i Stateflow Coder ® [10].
- Simular una xarxa amb un nombre significatiu de dispositius (en funció de la capacitat de processament). Un cop realitzat el model es pot encapsular i replicar per a formar una xarxa. Tot i que no es preveu en aquest treball, com a objectiu del projecte de recerca del grup s'inclou la creació tant de les capes físiques com del canal, és a dir, modelar la transmissió per l'aire i els obstacles que trobem, com ara parets, persones, etc.
- Al ser un projecte obert, tant en futurs objectius com en gent que hi intervindrà, les aportacions que s'hi facin s'han de poder adaptar a les possibles necessitats i mètodes de treball que es vagin requerint. Així es podrà acabar combinant des del nivell més alt, com aplicacions o estructures de xarxa, feina pròpia d'Enginyers Informàtics, fins al més baix nivell en que es troba el canal físic, per exemple experimentació de la transmissió a través de diferents tipus de substàncies tasca més pròpia d'un Enginyer en Telecomunicacions.
- L'entorn de simulació ha de ser el més genèric possible. Per a que usuaris de tots els àmbits es puguin involucrar en una futura evolució d'aquest projecte cal tenir present que les eines de treball han de ser molt flexibles, és a dir, han de donar els recursos necessaris per a desenvolupar qualsevol investigació o bé permetre interactuar, mitjançant interfícies, amb altres eines.

- Simulació hardware el més real possible. Per complir aquest requisit es disposen de dos alternatives amb una base comuna: el resultat d'ambdues s'ha de poder descarregar a un dispositiu, per exemple una FPGA (Field Programable Gate Array). Les alternatives són: (1) els models disponibles a l'entorn que provenen de llibreries que permeten treballar amb hardware específic, (2) l'ús de llenguatges de descripció de hardware, com VHDL. Amb aquesta estratègia es pretén eliminar deficiències com les comentades a l'apartat de les mancances dels actuals simuladors, veure apartat 1.2.2.
  La idea és poder aprofundir en el concepte de SDR, Software Defined Radio, és a dir, d'implementació flexible de protocols de radiofreqüència.
- Oferir una plataforma. Partint dels punts anteriors es pot acabar unificant el procés de disseny mitjançant MATLAB® i un Hardware basat en FPGA pensat per a SDR (alguns dels principals fabricants de FPGAs i DSPs també disposen de camins d'implementació des de MATLAB®).

Per a evitar confusions, la feina que es vol realitzar en aquest treball final de carrera és definir la viabilitat de la recerca mencionada tot implementant la part essencial de protocol: la capa MAC de l'estàndard IEEE 802.15.4, crear un model a nivells de MATLAB®, i arribar a simular-lo, a nivell de xarxa, utilitzant Simulink®.

1.3.2 Estratègia

Les decisions més rellevants que he pres (exceptuant les pròpies de la implementació que es detallaran a l'apartat corresponent) són dues: l'elecció del MATLAB® i la de Open-ZB del grup IPP-Hurray com a implementació de l'estàndard IEEE 802.15.4.

Dels entorns de desenvolupament detallats a l'estat de l'art, s'han descartat d'entrada els de pagament (dels quals no es disposa de llicència universitària). Dels restants, la majoria tenien mancances en quant a les eines i no permetien complir els objectius de crear una plataforma amb la qual es pugui treballar des de qualsevol àmbit. Un altre impediment ha estat que algunes es limitaven a la simulació sense permetre utilitzar el model per a generar el propi firmware, punt determinant a l'hora de seleccionar Simulink®.

Altres eines no han estat considerades perquè no permetien treballar amb llenguatge de programació C. Per contra, Simulink® ofereix diverses eines amb les quals es pot utilitzar codi C, entre les quals s'ha seleccionat Stateflow® . Els motius són: Les facilitats que ofereix per a intercanviar dades amb Simulink®, accepta variables globals a tot el model, el resultat és molt visual, i per tant senzill de comprendre per a futurs usuaris, i finalment la possibilitat de

depurar, almenys parcialment ja que permet veure l'evolució i les variables dels diagrames d'estat.

Una de les grans avantatges de MATLAB® és que és multidisciplinari, és a dir, podem simular codi C, via Black boxes o StateFlow®, descarregar-lo mitjançant codi genèric o directament als dispositius, com DSPs, emular hardware, per exemple FPGAs, simular blocs genèrics de processament, com up-converters o generadors de diferents tipus, així com processos físics (el canal) etc.

Finalment dos criteris decisius han estat: els coneixements previs de MATLAB® i Simulink®, que simplifica les primeres etapes d'interacció amb l'entorn, i que al ser eines populars existeix una comunitat d'usuaris important que pot ser de gran ajuda a l'hora de resoldre dubtes.

Pel que fa a l'elecció de la implementació de l'estàndard, primerament s'ha imposat el criteri d'independència de cap empresa, per diverses raons, principalment per disposar del codi font, evitar problemes de llicències i per no estar lligat a un hardware fixat per aquestes. També hem tingut en compte disposar d'un codi gratuït.

Un cop satisfets els primers criteris, dels restants s'ha seleccionat l'opció del Open-ZB, per tres motius: Primer, oferien la implementació de l'estàndard Zigbee (que pot ser interessant per a futurs treballs) per separat del 802.15.4; Segon, el codi està estructurat en arxius de forma que simplifica la tasca de localitzar cada part; i tercer el grup Hurray és molt actiu i accessible, la qual cosa facilita la possibilitat d'interactuar-hi.

Comentar per a acabar que s'ha seleccionat aquesta implementació tot i que està escrita en llenguatge nesC. Això suposa un impediment menor, ja que la principal diferència entre aquest i el C resideix en l'estructura, però les bases són les mateixes.

1.4 Estructura de la memòria

En aquest apartat es detallen les parts de que es composa aquest document.

Independentment dels propis capítols, com correspon a tota memòria de projecte, es comença amb l'índex del document i s'acaba amb el corresponent resum del treball.

En primer lloc es troba la introducció, on es posa en context el lector, amb algunes definicions i l'estat de l'art, i el més important es detallen els objectius del projecte. També es troba en aquesta secció els motius de les decisions relatives a l'elecció dels recursos utilitzats.

En segon lloc es presenta l'estudi previ, incloent tan l'estudi de viabilitat realitzat en un principi com les modificacions que han afectat al desenvolupament previst, un estudi de

costos, tot i que és només indicatiu, ja que en realitat no es pretén fer cap despesa real, i tan la planificació inicial com la planificació real al final del treball.

A continuació es troba el capítol dedicat a la implementació del projecte. En aquest es detallen les parts en que es divideix la feina, els principis de funcionament del sistema, com s'han dut a terme les diferents tasques, els diferents problemes trobats i les decisions preses per a solucionar-los i per últim un exemple del funcionament i els diferents tests que s'han pogut realitzar.

Per a finalitzar les conclusions amb idees per a ampliar-lo i les referències trobades en tot el document.

Amb aquesta informació s'espera donar una visió de la feina realitzada i de les possibilitats del resultat d'aquesta.

Capítol 2. Planificació i Viabilitat

En aquesta secció de la memòria, per una banda es presenta l'estudi inicial de la memòria i l'abast proposat, i per l'altra s'estudien les diferències i els canvis que ha calgut fer a la planificació a mesura que s'ha avançat en les diferents tasques per a obtenir l'evolució real del projecte.

2.1 Estudi de viabilitat

En aquest apartat es recupera l'estudi realitzat al començar el projecte, entregat com a part de l'informe previ el gener de 2008, i es compara amb el que es presenta en aquesta memòria, és a dir, s'avalua si he assolit els objectius marcats inicialment.

2.1.1 Entorn

Com reflecteixen els següents apartats extrets de l'estudi inicial des de bon començant ja queda patent que aquest projecte forma part d'un projecte de recerca major i més complex.

Projecte global

Avui en dia tenim sistemes de comunicació molt diferents. Dintre d'aquests podem distingir els cablejats dels sense fils. Aquest projecte es centra en l'àmbit dels segons. Els protocols que s'engloben en aquest grup són molt nombrosos, perquè no només es disposa dels protocols estàndards sinó també d'un gran nombre de propietaris, a més cadascuna de les capes en que es divideixen té disponibles diferents models, opcions, implementacions, etc. Aquest fet genera un seguit de problemes a l'hora de prendre decisions, des de quin dispositiu s'utilitzarà, fins a quin conjunt de capes serà el més adequat per a l'aplicació.

La solució que proposem és utilitzar els avantatges que ens dona un entorn de

simulació. L'objectiu del conjunt del projecte consisteix en desenvolupar un sistema que permeti triar cada una de les capes que intervenen en un procés de comunicació sense fils relacionar-les entre elles i simular el seu funcionament sobre una representació digital del canal de comunicacions que reflecteixi els inconvenients que es presentaran a la realitat i poder així veure com respondria el dispositiu. Un dels punts més importants d'aquest projecte és que ha de permetre simular un sistema "complet": des de la part hardware de transmissió i recepció (utilitzant tècniques de Software Defined Radio) fins la part software corresponent a l'aplicació de l'usuari. Per altra banda, es pretén treballar sobre l'entorn MATLAB®, la qual cosa permet utilitzar les parts simulades amb el dispositiu real, és a dir, agilitza la feina de l'usuari, que, un cop comprovat el funcionament, pot programar directament els dispositius des de MATLAB®.

Pel que fa a la gent que es dedica a la recerca pròpiament dita aquest projecte facilita moltes tasques tant als dissenyadors de hardware com als de software, perquè a uns i altres se'ls permet comprovar com reacciona la seva feina amb d'altres existents. També als que vulguin fer comparatives entre les diferents combinacions possibles tindran una eina amb la que treballar.

Tot i que a l'estudi previ els objectius van ser redactats des d'un punt de vista menys tècnic i entrant menys en detalls puntuals, és fàcil de veure que la intenció de la recerca és la mateixa. També cal tenir en compte que els propòsits es van dividir en dos apartats diferents, en el següent punt es troben els corresponents al projecte final de carrera.

Projecte propi

El projecte complert té un gran abast que no pot dur a terme una sola persona, no només per la quantitat de feina que suposa, sinó també degut a que es necessiten coneixements de diferents especialitats. Diversa gent hi treballarà realitzant diferents parts. La part que correspon al projecte final de carrera al qual aquest document fa referència abasta la implementació de la capa MAC del protocol IEEE 802.15.4, un dels més utilitzats per a les xarxes de sensors, i el desenvolupament d'una part de la capa de xarxa, com que el protocol IEEE 802.15.4 no té capa de xarxa, se n'implementarà una seguint els preceptes del protocol estàndard Zigbee que utilitza com a capa MAC el protocol anterior. Perquè aquests objectius s'integrin en el projecte complert s'haurà de desenvolupar sobre MATLAB® amb els avantatges i inconvenients que això suposa.

L'última part del projecte consistirà en testejar les implementacions realitzades amb la capa física del IEEE 802.15.4, realitzada per un altre membre del projecte, i posar a prova el funcionament.

En conclusió, l'objectiu d'aquest treball és, per un costat, obtenir un conjunt de mòduls MATLAB® que puguin ser combinats entre ells (i amb altres) per a obtenir una simulació d'un sistema de comunicacions; i per un altre, els mòduls obtinguts s'hauran de posar a prova conjuntament amb d'altres per intentar realitzar simulacions, inclús provar d'utilitzar-ho per a programar un dispositiu real.

Pel que fa als objectius particulars d'aquest treball es pot veure com s'han tingut que variar. Aquests canvis es van tenir que realitzar poc després de finalitzar l'estudi previ, els principals motius van ser la necessitat de finalitzar abans del previst, veure apartat 2.2.2, i un conjunt de problemes no previstos en relació amb la gestió de les dades i el context en que eren visibles, aquest i altres problemes apareguts es detallen a l'apartat corresponent 3.6. Com es pot veure en el punt dedicat a la planificació, veure apartat 2.2.2, aquests fets van provocar considerables variacions.

Els objectius van resultar en certa forma reduïts, però en previsió de que s'haurien de crear capes superiors per al test del protocol MAC es va decidir canviar la implementació del nivell de xarxa per una barreja entre aquesta i la d'aplicació que permetés posar a prova el projecte de forma més controlada.

Continuant amb l'estudi previ realitzat el següent apartat és el relacionat amb l'estat de l'art.

2.1.2 Estat de l'art

En aquest apartat s'han de considerar els diferents aspectes que intervenen tant en el conjunt com en la part corresponent al meu projecte.

Altres Simuladors

Aquest punt fa referència al projecte complert. En el mercat de les comunicacions sense fils existeixen simuladors que en principi poden semblar que realitzen tasques similars a la que es pretén desenvolupar, però si s'estudien més atentament es descobreix que implementen només algunes de les capes dels protocols que ofereixen. Per exemple, OPNET i OMNET només simulen la capa MAC i la de xarxa. A més, la majoria d'aquests programes no compilen els llenguatges de programació coneguts, sinó que són intèrprets del seu llenguatge particular, per tant s'ha d'implementar un codi només per a la simulació que no es podrà aprofitar. En canvi, el nostre projecte pretén no només permetre utilitzar llenguatges

com C o VHDL, i tots els llenguatges acceptats per MATLAB®, sinó que a més permetrà obtenir codi per al dispositiu de les parts no implementades per l'usuari.

Pel que fa a l'estat de l'art, cal comentar que durant el desenvolupament del propi projecte s'han conegut alguns simuladors més, aquest fet es veu reflectit en l'apartat corresponent, veure 1.2.3,d'aquest document.

Implementacions de les capes MAC i de xarxa

A diferencia de l'apartat anterior, aquest punt només influeix en el desenvolupament d'aquest projecte. D'implementacions de capa de xarxa n'hi ha diverses per a cada protocol, però on realment hi ha una gran diversitat és amb la capa MAC. En el cas del IEEE 802.15.4 companyies com TI o Freescale, entre altres, ofereixen gratuïtament les seves versions, però no ens són útils en cap dels casos pels següents motius:

- Encara que estan escrites amb llenguatges que el MATLAB® és capaç de compilar van lligades a un hardware específic del qual no ens interessa dependre.
- Es basen en sistemes de missatges o tasques que són complexes i no permeten independitzar les capes.
- No tenen en compte les particularitats de l'entorn MATLAB®.

A pesar de que la majoria de casos presenten els esmentats inconvenients se'n coneix un, Open-ZB que, tot i estar escrit en llenguatge nesC, està lligat a una plataforma senzilla i no compta amb complexos sistemes de pas de missatges o control de tasques, per tant pot ser útil.

En aquest apartat no hi han hagut canvis, quan es va realitzar l'estudi ja s'havien escollit els recursos. L'estat de l'art ha estat ampliat i és més específic, tant per la part d'entorns de simulació de models com per la del protocol.

2.1.3 Estudi de viabilitat

En els següents apartats extrets de l'estudi previ es discuteix la viabilitat del projecte des de diferents punts de vista, primer respecte als recursos, després com a projecte de recerca i per últim com a projecte fi de carrera (PFC).

Disponibilitat de recursos

Les eines necessàries per a desenvolupar aquest projecte són inicialment poques,

bàsicament el software MATLAB® i les seves eines Simulink® i Stateflow®. Amb aquestes ja en tenim prou per a implementar els mòduls de MAC i de xarxa, i generar una primera simulació. La universitat disposa de les llicències d'aquests productes, per tant, pel que fa a aquest aspecte, és viable. Per altra banda, a l'etapa de proves en un dispositiu real, es necessita la plataforma adequada per a descarregar-hi el programa generat per MATLAB® i també els mòduls necessaris per a interactuar amb aquest. Pel que fa a aquests dispositius es disposa d'un kit de desenvolupament de Renesas® de Zigbee i IEEE802.15.4 que permetrà testejar el funcionament. La plataforma en la que es pretén baixar el codi generat per MATLAB® és una de TI que incorpora FPGA, DSP i ARM, per tant suficient per a la tasca.

Els recursos discutits en el paràgraf anterior s'han mantingut, en el PFC no hi entra l'etapa de proves en entorn real, tot i així es disposa dels dispositius comentats i per tant també són viables per a la recerca.

Viabilitat tècnica

Es disposa d'uns objectius concrets i dels materials necessaris per a realitzar el projecte. Ara bé, MATLAB® ens donarà el rendiment suficient per a arribar a una simulació complexa? En principi aquest entorn ofereix potència més que suficient per aquesta tasca, encara que el projecte global acabi portant més al límit les seves possibilitats.

A més, es disposarà de suport tan pel que fa a l'entorn MATLAB® com pel que fa al protocol a implementar, aquest suport provindrà tan d'experts com de la documentació necessària, manuals o el document d'especificació de l'estàndard.

Actualment ja podem confirmar que MATLAB® ha donat el rendiment necessari per a simular la capa MAC amb un nivell superior i per a treballar amb dos dispositius, formats per aquestes capes, que es comuniquen entre ells. Tot i així, es mantenen els dubtes de quants dispositius es poden arribar a simular simultàniament, aquesta dada està lligada a la capacitat de l'ordinador sobre el qual es treballi, per tant no es disposa d'una dada determinada.

Viabilitat com a PFC

D'entrada recordar que no es coneix cap entorn de simulació que tingui en compte totes les capes dels protocols ni que permeti combinar-les, simular un canal de comunicacions físic i la possibilitat d'obtenir codi. Pel que fa al PFC, tot i que com s'ha explicat existeixen altres implementacions dels protocols, no se n'ha trobat cap que compleixi els criteris de MATLAB®.

També cal considerar que poden aparèixer problemes relacionats amb l'entorn com

ara que el compilador que ofereix no dona una interacció amb l'usuari massa clara, tan pel que fa a les opcions, com per l'especificació dels errors i warnings trobats. Per a aquests casos, s'ha previst treballar amb altres eines que permetin compilar i depurar errors de forma més clara per a, posteriorment, introduir el codi resultant a MATLAB®.

Els problemes previstos a l'apartat anterior han aparegut i ha calgut utilitzar alternatives per a solucionar-los. Les mesures preses han permès superar la situació. En l'estudi es comenten algunes possibles dificultats que, juntament amb les explicades a l'apartat 3.6, s'han anat resolent.

2.2 Planificació

Inicialment es va fer una planificació temporal, que difereix de la finalment realitzada en algunes de les tasques. A continuació es presentaran les dues i les raons per les quals s'han produït aquests canvis.

S'ha de tenir en compte que la dedicació és diària, però, degut a les hores que ocupa la feina, limitada a unes hores al dia. Aquest fet es va tenir en compte a la planificació.

2.2.1 Planificació prevista

A la figura 2.1 es presenta la que es va plantejar inicialment. Cal remarcar que l'organització del treball estava pensada per a ser finalitzat la setmana 28 cosa que no pot ser ja que l'entrega s'ha de produir abans de la setmana 25 (data que no es coneixia a l'inici).

En aquesta es poden distingir set tasques principals, les més importants i que es preveia que donarien més feina en aquest projecte eren previsiblement les capes MAC i la que

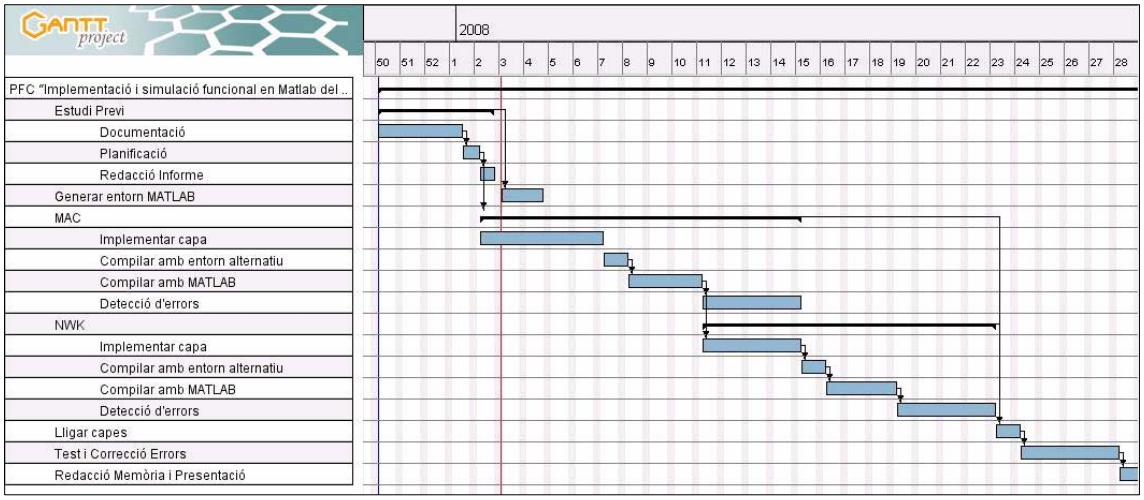


Figura 2.1. Planificació temporal del PFC

s'havia previst com a xarxa, per això es va preveure més temps per a aquestes. Com es pot veure aquestes tasques estaven divididas en les mateixes parts, la principal d'implementació i tot un conjunt de tasques complementàries que havien de servir per a la comprovació del funcionament i adaptació a l'entorn MATLAB®. Cadascuna de les tasques està descrita pel seu nom. Al següent apartat es fa una explicació més detallada de les que són més importants.

2.2.2 Execució real i desviacions

Estimar i preveure cadascun dels possibles problemes que poden aparèixer en un projecte és una tasca complexa, ja que desviacions respecte de les previsions són inevitables.

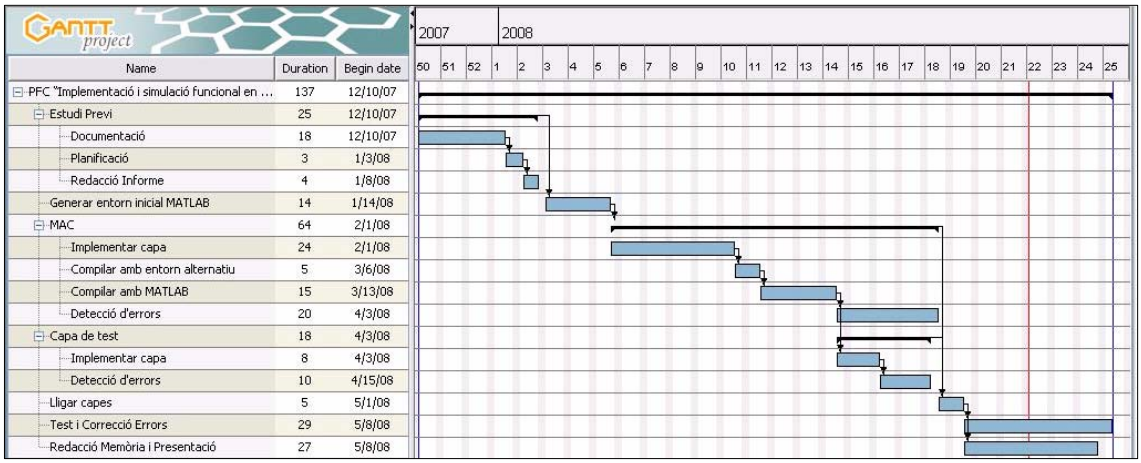
Comparant la planificació amb la figura 2.2, on apareix el seguiment real de la realització del projecte, es pot veure on apareixen problemes que provoquen retards, tot i que he pogut mantenir la seqüència de fases prevista.

Les qüestions personals també afecten el desenvolupament de la capa MAC. En rebre noticies relacionades amb les dates d'inici del curs 2008-2009, que realitzaré a l'estranger, que m'impedeixen defensar aquest treball més enllà del mes d'agost, per tant decideixo ajustar la planificació i els objectius per finalitzar el projecte dins les dates de la convocatòria de Juny. Com a conseqüència, redueixo considerablement la fase relativa a la capa de xarxa, eliminant algunes de les fases que ja no tenen sentit en el nou plantejament. Aquest és el canvi més important que ha succeït al llarg d'aquest treball.

Per últim una altra variació que s'ha produït ha estat la prolongació de l'etapa de test i correcció d'errors, això ha succeït degut a que el nombre de proves necessàries per a considerar correcte el funcionament de les capes és molt superior a l'esperat. Com a solució s'ha prolongat aquesta fase durant la redacció d'aquest document. Tot i que encara no es pot confirmar la conseqüència d'aquesta decisió, el més probable és un augment de la durada de la tasca de redacció.

Les fases que es considera que necessiten una explicació més detallada són:

- MAC → Implementar capa: Aquesta tasca està formada internament per dues parts que s'entrellacen. Una és la creació del diagrama d'estats en MATLAB® que realitza la capa MAC i l'altra és la conversió de llenguatge nesC a C, i introducció al diagrama de les parts de la implementació de la capa MAC de Open-ZB que m'interessen.
- MAC → Compilar en entorn alternatiu: es va preveure aquesta etapa com a conseqüència de la dificultat que suposa treballar amb el compilador de MATLAB®, que no detalla de forma clara alguns errors, de forma que



Name	Duration	Begin date
PFC "Implementació i simulació funcional en ...	137	12/10/07
Estudi Previ	25	12/10/07
Documentació	18	12/10/07
Planificació	3	1/3/08
Redacció Informe	4	1/8/08
Generar entorn inicial MATLAB	14	1/14/08
MAC	64	2/1/08
Implementar capa	24	2/1/08
Compilar amb entorn alternatiu	5	3/6/08
Compilar amb MATLAB	15	3/13/08
Detecció d'errors	20	4/3/08
Capa de test	18	4/3/08
Implementar capa	8	4/3/08
Detecció d'errors	10	4/15/08
Lligar capes	5	5/1/08
Test i Correcció Errors	29	5/8/08
Redacció Memòria i Presentació	27	5/8/08

Figura 2.2 Tasques i evolució temporal del projecte

compilaré (per solucionar els problemes que apareguin) amb un altre compilador de C.

- Lligar capes: En aquesta es creen subsistemes a partir dels blocs amb les diferents parts del protocol i es creen les connexions Simulink® necessàries.

2.2.3 Hores dedicades

Tenint en compte el nombre de crèdits que suposa un treball final de carrera la quantitat d'hores que ha d'involucrar el projecte gira en torn de les 375.

Per altra banda a la figura 2.2, l'evolució temporal, es calculen uns 137 dies sense tenir en compte els caps de setmana durant els quals també s'ha treballat, en total el nombre de dies és de 191. Aquest valor és poc concret perquè el nombre d'hores per dia varia. Durant la primera part del projecte no es va disposar de moltes hores per a dedicar a recerca, mentre que durant els últims mesos la dedicació ha estat molt més intensa. Fent un càlcul aproximat, comptant 1.5 hores diàries per a les etapes de menys dedicació i 3.5 per a les de més, resulten unes 2.25 hores de mitja. Això fa un total d'unes 430 hores, valor més que excessiu a les 375, però tenint en compte la complexitat del projecte s'opina que són les hores que corresponen per a completar els objectius.

Tot i que el nombre d'hores ha superat el recomanat es considera que són útils per a la recerca que es pretén continuar després de finalitzar aquest treball.

2.2.4 Estudi de costos

Com a projecte final de carrera no té molt sentit un estudi de costos degut a que tots els recursos que s'utilitzen, exceptuant els humans, són software lliure o bé software per al qual la universitat disposa de llicència. Per tant el cost del projecte és el cost en hores d'enginyer. Considerant un enginyer junior que cobra uns 25 €/hora fa un total de 10750€.

La despesa més important si no es consideren les llicències de la universitat és el MATLAB® i les seves eines. De fet aquest és l'únic cost que no es podria obviar en cas de no disposar de llicències perquè les altre eines o són gratuïtes, per exemple la de seguiment del projecte és el GanttProject té llicència GPL (gratuïta), o existeixen alternatives lliures, com el MS Office® que pot ser substituït pel OpenOffice.

Els costos per a una empresa, només del PFC, associats a l'ús del MATLAB® i les seves eines seria(cost industrial, els universitaris són menors),:

- MATLAB® 1950€
- Simulink® 3000€
- Simulink® Fixed Point ™ 1000€
- Fixed-Point Toolbox ™ 1000€
- Stateflow® 3000€

Amb les anteriors eines s'arribaria a un total de 9950€ i només permetria simular, ni generar codi ni treballar amb hardware, per a això farien falta altres eines. A més tampoc es disposaria de les toolbox necessàries per a implementar altres capes com la física o algunes aplicacions.

Al nivell de la recerca completa, els costos s'incrementarien de forma important perquè també s'haurien d'incloure diferents dispositius, alguns de senzills, simplement per a provar que la comunicació és possible encara que tinguin implementacions dels protocols diferents, i altres complexes, preparats per a SDR.

2.2.5 Conclusió del estudi

Per a finalitzar, a l'estudi es va incloure una petita conclusió sobre com es veu la viabilitat del projecte. La conclusió va ser:

Resumint els apartats anteriors es pot determinar que es tenen clars els objectius, s'està innovant en l'aspecte de que no s'ha trobat cap entorn de simulació igual ni cap protocol com el que es pretén implementar per a MATLAB®, és un projecte viable tan pels materials com pels possibles problemes que es consideren que poden anar apareixent i finalment sembla possible acabar-lo amb un temps raonable.

Per tant podem concloure que el projecte és viable i es pot començar-hi a treballar.

La conclusió de l'estudi no va ser equivocada encara que les consideracions respecte el temps deixen de tenir sentit, ja que la planificació va canviar. A pesar d'aquest canvi en cap moment s'ha dubtat de que en més o menys temps el projecte era viable. Si es tinguessin en compte els costos de sou i eines aleshores la viabilitat ja no és tan clara i depèn de molts altres factors, com ara l'empresa o centre de recerca per al qual es realitzés i l'interès que aquest tingués en els resultats.

Capítol 3. Desenvolupament

Per a explicar els resultats del projecte es començarà tractant les alternatives que s'han plantejat i descartat, tot seguit es farà una introducció al mètode seleccionat per a ajudar a entendre les implicacions d'aquest. A continuació es detallaran cadascuna de les parts seguint l'ordre en que es va implementar. Per a ajudar a la comprensió del treball complet es reproduirà un exemple. Finalment es llistaran els problemes més importants que s'han trobat i les solucions que s'han aplicat.

3.1 Alternatives

Un cop s'ha seleccionat Simulink® encara queda una decisió important a prendre, quin dels diferents mètodes d'implementació es seguirà?

Entre els disponibles n'hi han 3 que en un principi semblen vàlids, entre ells el finalment seleccionat.

El primer que es va considerar van ser les S-Functions, a través del bloc S-Function builder que simplifica la utilització d'aquestes. Aquestes consisteixen en un mòdul de Simulink® en el qual s'hi pot introduir codi en llenguatge C, a més proporciona la interfície d'entrada/sortida per a intercanviar dades amb el Simulink®.

Aquest genera un codi C equivalent combinant el codi donat amb el que implementa ell mateix, a més crea els "wrappers" necessaris per a poder-lo utilitzar des de Simulink®.

En principi aquest semblava vàlid, però té un seguit de problemes que fan que ens resulti inútil. El primer és que executa el codi que té internament de forma completa sense actualitzar les sortides fins que arriba al final, això fa que se'n puguin perdre variacions perquè no s'arribin a actualitzar mai. Aquest fet afecta principalment quan es vol interactuar amb parts reproduïdes directament al Simulink®, com ara *timers*.

El segon és que no accepta funcions internes, és a dir, cada S-Function és una funció, per tant s'hauria de generar un bloc per a cadascuna, però això ens porta al següent problema:

no es poden generar variables globals entre aquestes funcions, en conseqüència s'hauria de passar una gran quantitat de dades en forma de senyal de Simulink®, cosa que no té molt sentit. Per aquests motius es va preferir optar per una altra opció.

La següent és oferta per l'empresa Xilinx®, aquesta està orientada a les FPGAs. La solució consisteix en un mòdul Simulink®, anomenat PicoBlaze, que és l'equivalent a un processador de 8 bits que es pot descarregar en una FPGA. Aquest funciona afegint un mòdul de memòria amb el programa que ha d'executar. El problema bàsic que provoca que aquesta opció sigui descartada és que el nombre d'instruccions màxim que pot arribar a tenir en aquesta memòria és de 1024, degut a la mida del bus de direccions. Això és insuficient per a la capa MAC.

També existeix una versió superior a l'anterior, el MicroBlaze, que modela un processador de 32 bits, tot i així aquest no es va ni considerar degut a que no existeix la possibilitat de simular-lo, només està disponible per a descarregar-lo.

L'opció de treballar amb Stateflow® no només evita alguns inconvenients, sinó que també és més visual i facilita la comprensió a futurs usuaris o desenvolupadors. Una altra avantatge que té respecte els anteriors és que permet depurar, tot i que no permet entrar a la part de codi C, només permet veure com s'avança dins el diagrama d'estats.

3.2 Stateflow®: bases de funcionament

El Stateflow® és una eina per a crear autòmats finits, a més permet incloure codi en llenguatge C a diferents punts del diagrama d'estats. En aquest apartat s'expliquen les opcions que ofereix i com funciona. Primer s'expliquen els elements que s'han utilitzat, tot seguit com actua de forma conceptual i per a acabar com opera internament.

Abans de començar, mencionar que cada instància de Stateflow® pot ser un "Chart" o bé una "Truth Table", veure imatge 3.1, com el segon no s'ha utilitzat només s'explica el relatiu als primers.

3.2.1 Elements

Les parts que formen un diagrama d'estats són nombroses degut a que no només es disposa de les típiques, les més utilitzades són:

- Estats: Són el principal element, sense aquest no hi ha diagrama. Estan representats per un rectangle ovalat en els angles, veure figura 3.1. En aquests succeeix el més important: accepten codi en llenguatge C i crides a funcions externes. Una unitat de temps es comença en un estat i s'acaba en un estat, sigui el mateix o no.

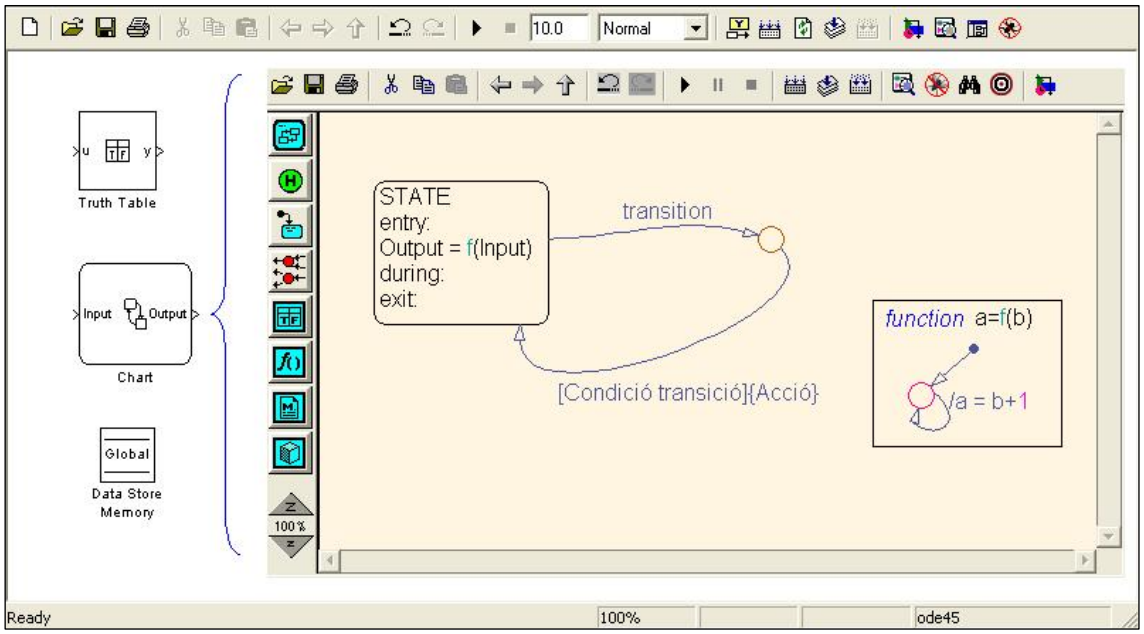


Figura 3.1 Elements de Simulink® i Stateflow®

- Transicions: Es mostren com a fletxes amb un destí i normalment un origen, tot i que no és necessari, veure figura 3.1. A cada instant n'hi pot haver una i només una activa, però en una unitat de temps es pot passar per totes les transicions que siguin necessàries. Aquestes poden tenir associades condicions que determinen la seva activació i també accions relacionades amb la condició o amb la transició pròpiament.
- Junctions: Aquests es representen amb petits cercles. Aquests no poden tenir ni condicions ni accions associades. Tot i així són molt necessaris perquè permeten encadenar transicions, és a dir, són punt d'origen i de destí d'una o vàries transicions. Al no ser estats no estan lligats al temps, per tant en una unitat de temps es pot passar per molts.
- Functions: Aquests elements són quadrats que engloben transicions i junctions. Tal com el seu nom indica actuen com a funcions, és a dir, s'instancien una vegada de forma independent a la resta del diagrama i poden ser cridades tantes vegades com sigui necessari des d'estats o transicions. Aquestes s'executen de forma completa dins de la unitat de temps, per aquest motiu no permeten estats al seu interior.
- Altres elements: A més dels anteriors es disposa de més instruments per a treballar, però no han estat necessaris en aquest projecte. Alguns exemples són la "truth table", les funcions MATLAB® o les history junctions.

3.2.2 Variacions als estats

Els estats permeten més que executar un conjunt d'instruccions, els estats es poden combinar. És a dir, un estat pot englobar altres estats i d'aquí apareixen els subsistemes.

Què s'aconsegueix amb aquestes combinacions? D'entrada permet encapsular, i per tant no només aïlles parts sinó que també divideixes la complexitat. Però la característica més important no és aquesta, sinó la possibilitat de paral·lelitzar l'execució, és a dir, Stateflow® permet generar estats en els que hi pots incloure'n d'altres que treballin simultàniament, veure figura 3.2. Això permet controlar i decidir diferents accions a cada unitat de temps. Lògicament no és que MATLAB® aprofiti els recursos dels processadors per a executar literalment en paral·lel, sinó que els executa seqüencialment, però dins d'una unitat (o període) de temps, és a dir, és un mètode per a trencar el fet que només es pugui anar d'un estat a un altre en una unitat de temps.

Com s'explicarà en els següents apartats, especialment veure 3.3.4, s'ha tret rendiment d'aquestes propietats sobretot a la part de control del *timer*, ja que permet simultàniament considerar les interrupcions a generar, les confirmacions a les interrupcions i la pròpia gestió dels comptadors.

Per altra banda també convé explicar que els estats divideixen les accions que inclouen en tres parts. L'usuari ha d'especificar a quina de les tres vol incloure cada una de les instruccions, això es realitza mitjançant unes etiquetes.

La primera part, ***entry***, inclou totes les accions que s'han de realitzar només quan s'activa l'estat, és a dir, just abans de la finalització de la unitat temporal. La segona, ***during***, engloba les que es poden executar varies vegades, però només una vegada per a cada unitat de temps, per tant són les que es van realitzant repetidament mentre no es surt de l'estat. Per últim ***exit***, aquest es bastant similar al primer, però quan es surt de l'estat. En conseqüència és el primer que es processar quan s'inicia la unitat de temps.

La combinació dels anteriors permet sincronitzar diferents accions de diferents diagrames i és molt important tenir-les en compte. A la pràctica ***entry***, ***during*** i ***exit*** són etiquetes que es col·loquen als estats prèviament a les instruccions d'aquest.

3.2.3 Tipus de dades i àmbits d'existència

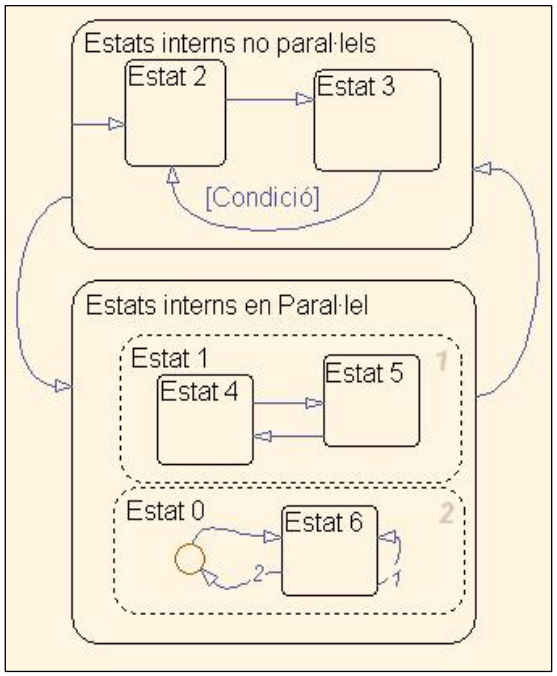


Figura 3.2 Combinacions d'estats

Stateflow® reconeix tipus de dades enters amb signe o sense i de mides de 1, 8, 16 o 32 bits. A més d'aquests també permet singles i doubles. Però com s'explicarà a l'apartat 3.6, un dels problemes és que no reconeix els punters, que tot i no ser un tipus pròpiament és molt necessari. En aquest apartat es presentaran els existents.

Dins d'un diagrama d'estats es permeten definir 5 tipus de dades depenent de la seva funció, es té: ***entrades, sortides, locals, constants, paràmetres*** i per acabar una forma de globals anomenats ***data store memory***. ***Entrades i sortides*** tenen la seva representació externa al bloc que representa la ***chart***, mentre que la resta no. El tipus més particular és ***data store memory*** que permet accedir a una variable que ha d'estar definida a Simulink® mitjançant un bloc específic per a aquest propòsit. Tot model que comparteixi el nivell amb aquest bloc, veure figura 3.1, o que estigui per sota, és a dir, que estigui inclòs als anteriors, té la possibilitat de llegir o escriure en aquesta variable.

Per altra banda, Stateflow® també permet treballar amb dades no definides als diagrames, ni tan sols a MATLAB®, sinó que siguin del codi que s'inclou i es declarin com a externes. Això suposa una gran avantatge a l'hora de compartir informació entre el llenguatge C i l'autòmat. Tot i que això ha de simplificar el desenvolupament de forma important, de fet no seria possible sense aquesta possibilitat, també ha suposat molts problemes principalment per l'aspecte comentat al principi sobre els punters, i també perquè el tractament dels vectors no és el que es pot esperar d'entrada. Aquests problemes es detallen a l'apartat corresponent, veure 3.6. Pel que fa al pas de dades entre aquests ha estat necessari un rigorós control per a comprovar que el funcionament era el desitjat. Això ha obligat en molts casos a examinar el codi generat per Stateflow®, és a dir, el codi equivalent a l'autòmat finit.

3.2.4 El resultat

Un cop finalitzat el disseny, inclosos els fitxers externs i afegits els components de Simulink® necessaris ja es pot fer el "***Build***". Això consisteix en generar l'equivalent del diagrama en codi C, compilar-lo i generar un conjunt d'arxius que MATLAB® utilitzarà per a la simulació. Els arxius resultants es poden examinar i són especialment útils els del llenguatge C perquè permeten entendre el funcionament del diagrama, com es passen les dades entre el propi codi i el de Stateflow® i localitzar errors més fàcilment.

Una de les particularitats d'aquests blocs, ***charts***, és tot i que pots crear varies instàncies d'aquest totes estan relacionades internament, és a dir, comparteixen el codi de l'usuari i les variables que hi hagi en aquest. Aquest fet pot ser una avantatge en determinades situacions, però en aquest cas ha resultat problemàtic, veure apartat 3.6.

Pel que fa a l'execució, a cada unitat de temps de Simulink® es segueixen els següents passos. Per a la primera instància d'aquest tipus de bloc es capturen les dades d'entrada, s'executa la part de l'estat actiu corresponent a l'etiqueta "***during***", tot seguit s'avaluen les transicions de sortida i si per a alguna es compleix la seva condició s'activa, prèviament s'executa la part de l'estat relacionada amb l'etiqueta "***exit***". A partir d'aquest moment es va passant per les transicions necessàries, realitzant les instruccions associades, fins a arribar a un altre estat, per a acabar s'activa i es duen a terme les accions que indiqui l'etiqueta "***entry***". Això es repeteix per a cada instància de Stateflow® i dins d'aquesta per a cada conjunt d'estats en paral·lel, veure imatge 3.2.

3.3 Desenvolupament: pas a pas

En aquest apartat és a on s'explicarà el que s'ha fet, per a fer això es seguirà l'ordre en que es van anar realitzant les diferents fases per les quals s'ha passat. Les primeres són les relatives al codi utilitzat de Open-ZB, degut a que aquest s'havia d'entendre i tractar. Les següents ja són part de la creació de l'autòmat i de les diferents parts de suport implementades amb mòduls Simulink®.

3.3.1 Estudi de Open-ZB

Un cop es va triar aquesta implementació de la capa MAC es va començar la lectura de la part de la documentació que en feia referència [14]. D'aquesta es va treure informació important relacionada amb el funcionament del protocol. Tot seguit es va procedir a separar les funcions que clarament serien útils, per a fer tot això es va realitzar tot un conjunt de documents amb aquesta informació. Alguns exemples d'aquestes són les que creen els

paquets, les que n'extreuen les dades o les de les interrupcions de sincronització, entre altres. S'ha de dir que es va intentar aprofitar al màxim les parts ja implementades, tot i que això va suposar una major feina en quant al tractament d'aquestes funcions.

El següent pas va esser un anàlisi de les relacions, és a dir, extreure quines funcions necessiten les triades i aplicar això de forma recursiva fins a tenir totes les dependències localitzades. En aquest pas és quan es va veure que seria inevitable utilitzar altres diagrames per a realitzar tasques de suport per al control de les sincronitzacions, les interrupcions i els *timers* associats a aquests.

Per a realitzar tot aquest procés es va utilitzar l'editor UltraEdit-32, un cop seleccionades les funcions, aquestes es van copiar en un document de text, es va agafar cadascuna d'aquestes i es van afegir al document les que es cridaven des d'aquesta. A mesura que s'avançava cada vegada es copiaven menys funcions, o perquè ja hi eren o perquè no hi havien crides, fins que es va finalitzar aquesta tasca. Per a trobar totes les relacions es va haver de buscar en diferents arxius, això va ajudar a determinar l'origen i el destí d'algunes de les funcions.

El següent pas va ser eliminar les parts que no resultaven útils i agrupar les funcions per grups segons les relacions. D'aquesta part restant es va examinar l'origen i el destí de cadascun dels grups i per a cadascun en va resultar un camí entre entrades per la capa física fins a sortides per la capa d'aplicació o a l'inversa. No en tots els casos la relació era tan directa, però sí en la majoria, per exemple algunes que són cridades des del nivell d'aplicació simplement varien paràmetres de la pròpia capa MAC.

En la majoria de casos això es va dur a terme començant amb les funcions que es cridaven des del propi nivell, però que no estaven implementades en aquest, és a dir, les que servien d'accés a altres nivells. A continuació es feia una cerca de les crides a la funció que contenia la primera i així successivament fins que s'arribava a una altra que tampoc pertanyia al nivell. Per a clarificar la idea un exemple senzill seria el següent:

Començant per una crida des de MAC a la capa superior ***MCPS_DATA_indication*** que es trobava al procediment ***indication_data***, es procedeix a buscar aquest últim i el resultat és que es crida des de ***data_indication*** que alhora prové de ***PD_DATA_indication*** i aquest ja té els precedents a la capa física per tant ja hem lligat l'entrada i la sortida. Si en algun dels passos existeix més d'una crida es tenen totes en compte.

Aquest procés es va realitzar amb més de 25 grups arribant a un total de més d'un centenar de funcions.

El següent pas va esser completar els forats que quedaven en la documentació generada, és a dir, es van afegir les parts que havien de ser pròpies del diagrama, tot i que només es va posar la idea.

Degut a que a la implementació en llenguatge C s'utilitzen tipus de dades complexos, tipus estructures o enumerats, que no tenen representació en Stateflow® s'ha procurat deixar les parts que tracten aquest per a ser compilada directament del codi C i simplement realitzar crides des del diagrama d'estats.

Pel problema anterior i perquè el propi disseny dels objectes i l'espai de l'entorn no ho permeten la major part de tasques estan en fitxers separats implementats amb llenguatge C. Com es pot veure al diagrama, veure apèndix 1, els espais que estan disponibles no s'hi pot afegir grans quantitats de codi.

Un cop es va arribar a aquest punt es va tenir tot el projecte de forma conceptual en la documentació creada.

3.3.2 Conversió del llenguatge de programació

Un cop es van seleccionar les parts que es volien utilitzar va ser necessari canviar-les de llenguatge de programació. Tot i que el nesC i el C són similars tenen algunes diferències importants que es van tenir que canviar.

El nesC està pensat per a treballar sobre el sistema operatiu TinyOS mentre que C és més simple en aquest aspecte.

En la majoria dels casos va acabar essent necessari simplement eliminar estructures o paraules clau, a continuació es detallen les accions realitzades més rellevants.

L'acció més significativa va ser la substitució del camp ***Provides*** i del ***uses*** pels ***includes*** propis de C. En conseqüència es van tenir que crear els fitxers de capçalera amb les definicions de les funcions i de les variables que s'accedien des d'altres fitxers.

En nesC s'utilitza la paraula clau ***atomic*** per a definir operacions o conjunts d'aquestes que han d'executar-se de forma continua, sense interrupcions. En aquest cas es van substituir per una crida a una funció ***inline*** en la qual es pot col·locar les instruccions necessàries per a aturar les interrupcions, de moment no se n'utilitzen degut a que no hi han interrupcions a la simulació, a més aquestes han de ser específiques del hardware amb el que es treballi.

Les comandes que es van eliminar són ***call*** i ***signal***, aquestes serveixen per a cridar a ***commands*** i ***events*** respectivament que en C no deixen de ser funcions com la resta. També es van suprimir paraules clau del compilador com ***norace*** o ***async***, aquesta última serveix per a indicar que una funció pot ser cridada des d'una interrupció. A més en nesC també hi ha un tipus de procediments marcats com a ***tasks***, aquests són com la resta però es gestionen diferent des del sistema operatiu, i que es criden amb l'operació ***post***, també eliminada.

Un altre detall important és que per a certes funcions es va tenir que realitzar una cerca des de la capa MAC fins a un origen indeterminat, aquestes últimes eren funcions associades a una interrupció, però com que a nesC no s'especifica, exceptuant la paraula clau ***async*** que també s'utilitza a altres llocs, va ser gràcies a la documentació que es va solucionar aquesta incògnita.

En tots els casos anteriors es van considerar les conseqüències dels canvis realitzats, que en la majoria no va suposar un problema degut a que eren per a controlar detalls del compilador o per a gestionar com s'executen algunes tasques sobre el sistema operatiu. Al no considerar-lo no hi ha inconvenient en suprimir-los.

Aquesta tasca es va dur a terme amb l'editor UltraEdit buscant les paraules clau, prèviament es van repassar els codis en busca d'aquestes diferències i mitjançant manuals de nesC [15] es va decidir quina acció dur a terme.

Per últim es va utilitzar un compilador de C estàndard, en aquest cas per un microcontrolador 8051, per a comprovar si hi havien errors i solucionar-los. Per a poder realitzar aquesta tasca es van tenir que generar fitxers amb funcions que substituïssin les cridades dels nivells superior i inferior, posteriorment aquests no van ser necessaris perquè van ser canviats per entrades i sortides del Stateflow®.

Un cop es va poder compilar sense errors es va donar per finalitzada aquesta tasca, en aquest punt ja es disposava de la implementació de les funcions en llenguatge C. Tot i així, es van tenir que realitzar modificacions per a poder adaptar-les al diagrama d'estats.

3.3.3 Autòmat finit, la capa MAC

La creació del diagrama va ser la part més complexa, perquè va ser necessari tenir en compte molts aspectes. Primer es procedirà a una explicació del conjunt i de cadascuna de les parts, posteriorment s'explicaran alguns detalls de la implementació. Finalment també es comentaran altres ***charts*** creades per a donar suport a la principal. Si es vol una descripció funcional veure l'apartat 3.5.

A la figura 3.3 es pot veure el resultat complert, degut a la mida no es poden apreciar els detalls, però a l'apèndix 1 s'inclou una imatge més gran.

El funcionament general és el següent: s'inicia per l'estat de mida major anomenat ***State_MAC_enter***, en aquest s'inicien totes les variables i les sortides, a continuació depenent de les entrades es selecciona el camí que es seguirà. Això està implementat com una vintena de ***junctors*** que es troben per sota i a l'esquerra de l'estat comentat. En aquesta fase en que es

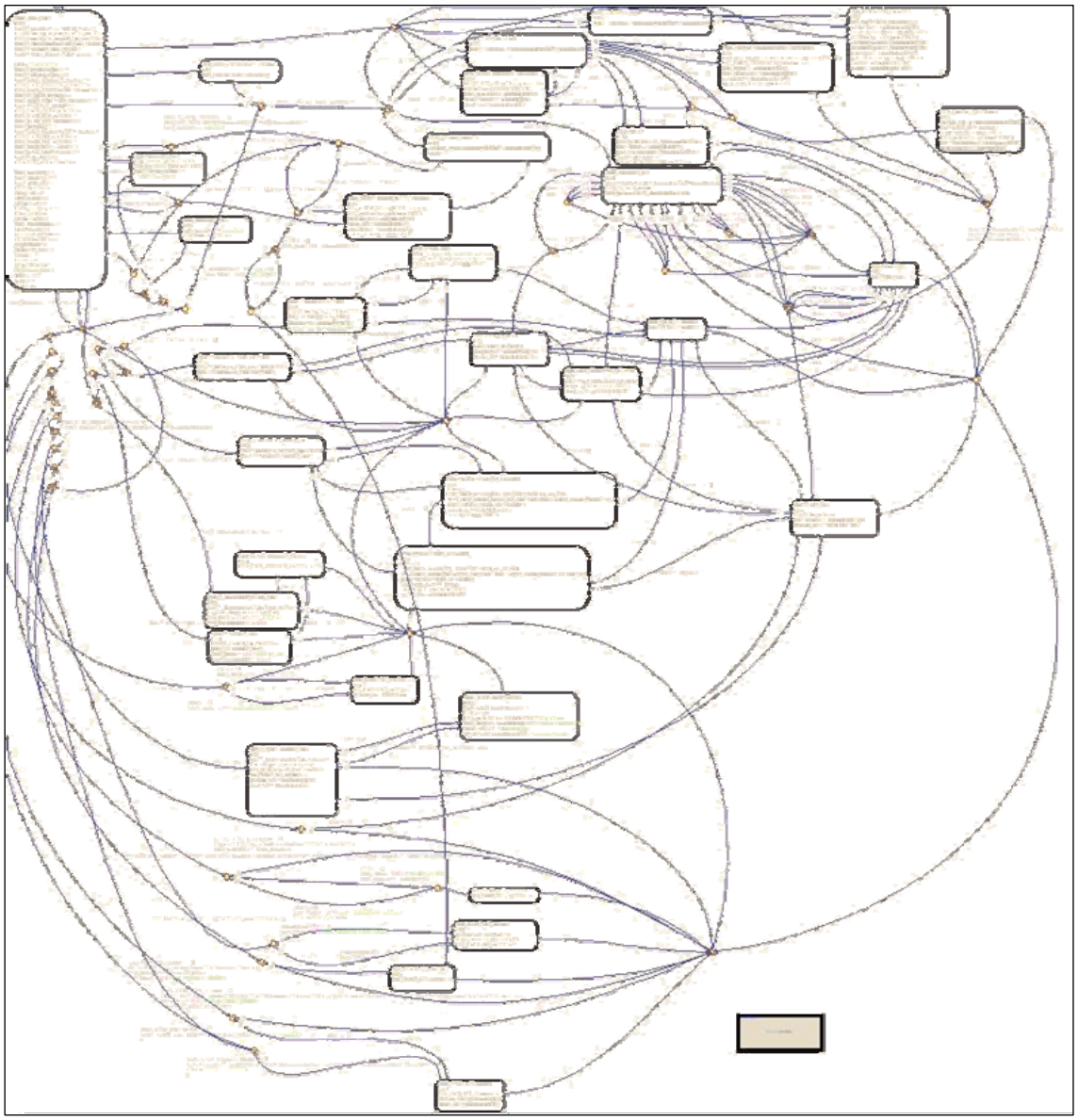


Figura 3.3 Autòmat finit que implementa la capa MAC.

tria el camí és quan es tenen en compte les prioritats i les sincronitzacions, aquests conceptes s'expliquen posteriorment en aquest apartat. Un cop s'ha activat la transició corresponent a l'entrada activa es van succeint estats i transicions fins arribar a alguna variació de les sortides o bé es torna a l'inicial. En qualsevol cas sempre hi ha un camí cap a l'inici que es segueix abans o després de provocar alguna acció.

Aquest cicle es va repetint ininterrumpidament des de que s'inicia fins que s'acaba la simulació.

En gairebé tots els estats i les transicions es realitzen accions i s'avaluen condicions que porten a determinar quin serà el següent camí a seguir. Totes aquestes operacions estan implementades en llenguatge C i al diagrama hi apareixen les crides corresponents, el valor de retorn de les quals és el que indica al Stateflow® cap a on ha d'evolucionar la simulació.

El diagrama es pot dividir en parts a on cadascuna té una funció diferent. A continuació s'explica què fa cada part i com ho fa.

- Inicialització. Quan es vol començar a treballar amb la capa d'accés s'han de realitzar dos operacions. La primera és donar l'ordre corresponent a els *timers* i les funcions de suport encarregades de les sincronitzacions, veure part superior de la figura 3.4. La segona és iniciar alguns paràmetres amb els valors de configuració desitjats, veure la figura 3.5. A diferència de la primera part aquesta última pertany a l'estàndard 802.15.4.

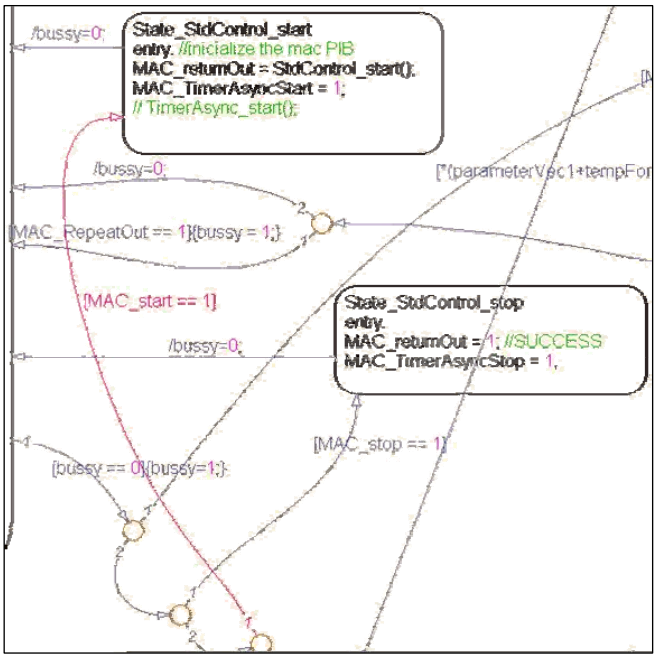


Figura 3.4 Secció d'inici i final del nivell MAC

A la imatge 3.4 es pot veure l'estat ***State_StdControl_start***, el primer a ser activat, aquest crida a la funció ***StdControl_start*** i activa la sortida ***MAC_TimerAsyncStart***, entrada d'una altra instància de Stateflow®, la ***TimerAsync*** que s'explica més endavant en aquest apartat.

L'acció que es duu a terme a la transició amb la condició *MAC_MLME_START_request == 1*, segona fase de la inicialització, succeeix gairebé tota dins del codi cridat per la funció ***MLME_START_request***, però el

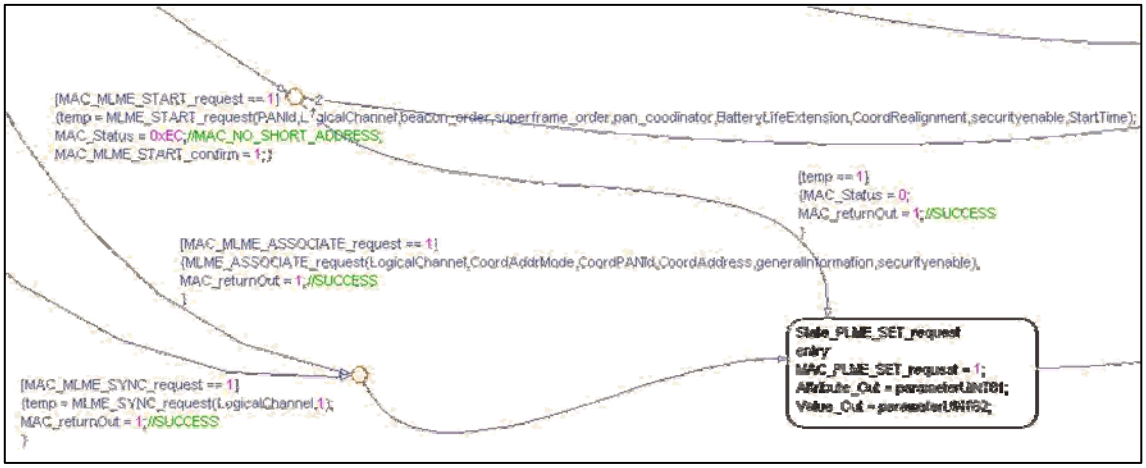


Figura 3.5 Operacions associades a les primitives Start, Associate i Sync de l'estàndard

resultat d'aquesta es recull com el valor de retorn i determina si es passa a l'estat ***State_PLME_SET_request*** o bé si no es realitza res més. En qualsevol dels casos sempre s'activa la sortida ***MAC_MLME_START_confirm*** per a informar a les capes superiors que ja s'ha processat la petició i amb MAC_Status s'informa del resultat en cas d'error, en cas d'èxit ***MAC_returnOut*** passa a valer '1'.

La part d'aturada és molt més simple, l'únic que es fa és indicar al ***chart TimerAsync*** que pari d'enviar interrupcions i confirmar la petició.

- Scan. La petició per a iniciar el procés de ***Scan*** és una transició en la qual es crida a la funció ***MLME_SCAN_request***, que calcula els valors d'algunes variables, i es prepara la sortida ***MAC_genericOut*** amb el valor que indica quin *timer* s'ha d'activar, d'aquest valor depèn quina interrupció saltarà. Segons el resultat de la funció es torna a l'inici o bé s'activa l'estat ***State_Timer_start*** que posa a la sortida els valors de configuració d'aquest i inicia el procés posant a '1' la sortida ***MAC_Timer_start***.

- Estats associats a les interrupcions produïdes pel diagrama de suport ***TimerC***. Hi ha tres tipus de *timers* cadascun genera una interrupció diferent. Si s'observa la figura 3.6, l'estat superior correspon a l'entrada ***MAC_T_ResponseWaitTime_fired***. Aquesta s'utilitza quan es fa una petició i espera la resposta fins un temps màxim. Per exemple, s'utilitza al intentar associar-se, tot i així s'activa el *timer* quan es rep l'*acknowledge* de la petició.

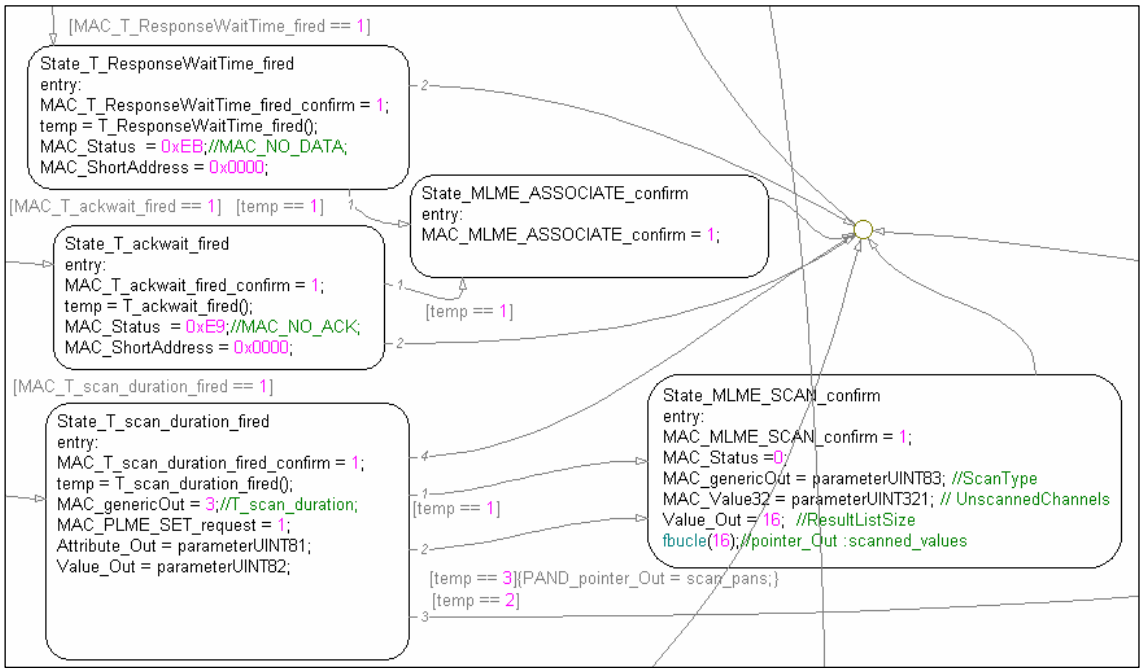


Figura 3.6 Estats associats amb les interrupcions generades pel diagrama TimerC

Quan s'activa aquest estat vol dir que el temps ha estat superat i no es té resposta, l'acció que duu a terme en aquest cas és activar igualment la confirmació de la petició, però amb la variable ***MAC_Status*** indicant que s'ha produït un error.

La següent interrupció és ***T_ackwait_fired*** i simplement s'activa cada vegada que s'ha realitzat un enviament amb el camp de confirmació seleccionat, per tant s'espera un *acknowledge*, i no es rep en el temps esperat. Si s'activa i no ha arribat al nombre màxim de reenviaments inicia el procés per a tornar-lo a enviar. En el cas oposat, elimina el paquet de la cua, si es tractava d'alguna petició que esperava confirmació respon a la capa corresponent l'error

Per últim la ***T_scan_duration_fired*** és la que activa l'estat que realitza el ***scan***. Cada vegada que es crida a la funció s'incrementa el canal en que es fa el ***scan*** fins que s'arriba a l'últim moment en que es genera la confirmació amb la informació recollida i s'entrega a la capa superior. Per a fer això s'utilitza el retornat per ***T_scan_duration_fired()*** i/o es torna a activar el *timer* corresponent, aquesta opció no es veu a la figura 3.6, o bé s'activa l'estat ***State_MLME_SCAN_confirm*** i s'actualitzen les sortides amb la informació obtinguda.

- Associació. La part del diagrama encarregada d'iniciar el procés d'associació d'un dispositiu a la xarxa es redueix a una simple transició, la qual es pot observar a la figura 3.5, que s'activa quan des de la capa superior es posa a '1' l'entrada ***MAC_MLME_ASSOCIATE_request***. En realitat aquest procés és més complex del que es descriu aquí, només s'està explicant la petició, el funcionament està més detallat a l'exemple, apartat 3.5. Mitjançant la funció ***MLME_ASSOCIATE_request***, els paràmetres de la qual provenen tots de les entrades del diagrama, es crea el paquet corresponent i es posa a la cua de paquets per a enviar, igual que en el cas anterior també es confirma que s'ha atès la petició.

  Com es veu a la imatge 3.5, la següent transició és incondicional, per tant sempre es finalitza passant a l'estat ***State_PLME_SET_request***, serveix per a enviar comandes a la capa física de canvi de configuracions, i mitjançant les variables globals ***parameterUINT81 i 2*** s'estableix quina variable canviar i per quin valor. Per exemple en aquest cas des del codi C es demana que la radio passi a estar en el canal indicat com a paràmetre. Finalitzat aquest es torna a l'inici.

- Interrupcions de sincronització. En aquest punt englobem totes les que provenen de la instància de Stateflow® ***TimerAsync***. Aquestes interrupcions assenyalen a

quin punt del ***superframe*** estàs, i per tant, quines tasques has de dur a terme. Com es pot veure a la figura 3.7 es tracten 3 interrupcions: ***bi_fired***, ***time_slot_fired***, ***backoff_fired***. També es pot observar que n'existeix una quarta ***sd_fired***, aquesta es gestiona internament al codi cridat des del diagrama ***TimerAsync***.

Per a entendre el que fa cadascun, s'ha d'entendre l'estructura del ***superframe***. Aquest consisteix en una forma de divisió temporal. Les parts que el formen són dues: una activa i una inactiva. La part activa, que es divideix en Contention Acces Period (CAP) i Contention Free Period (CFP), comença amb l'enviament d'un *beacon*, la resta de temps es divideix en 16 parts anomenats ***slots***. Si formen part del CAP, estan alhora dividits en tantes parts com sigui possible. Depenent del temps actiu restant, els dispositius intenten utilitzar el canal, seguint el Carrier Sense Multiple Access with Collision Avoidance(CSMA-CA). Si formen part del CFP aleshores els ***slots*** estan dedicats a un sol dispositiu, Guaranteed Time Slot (GTS), i no es necessita utilitzar CSMA-CA.

Exceptuant les peticions que l'estàndard especifica que s'han de confirmar, les úniques que tornen aquest tipus d'informació són les interrupcions. A més, a diferència de les primeres, el senyal d'interrupció no es desactiva fins que es rep aquesta confirmació. Es va decidir així degut a que és el procés que segueixen els microcontroladors que acostumen a gestionar aquest tipus de protocol, veure apartat 3.6.

Seguint l'ordre dins del ***superframe***, el primer que s'activarà serà l'estat ***State_TimerAsync_Bi_Fired*** en aquest es confirma la interrupció i es crida la funció ***TimerAsync_bi_fired***. Si es tracta del dispositiu coordinador de la PAN, enviarà un *beacon*. Per a fer això es còpia la longitud del paquet de ***parameterUINT81*** a la sortida ***psdu_length***, es posa el paquet a la sortida,

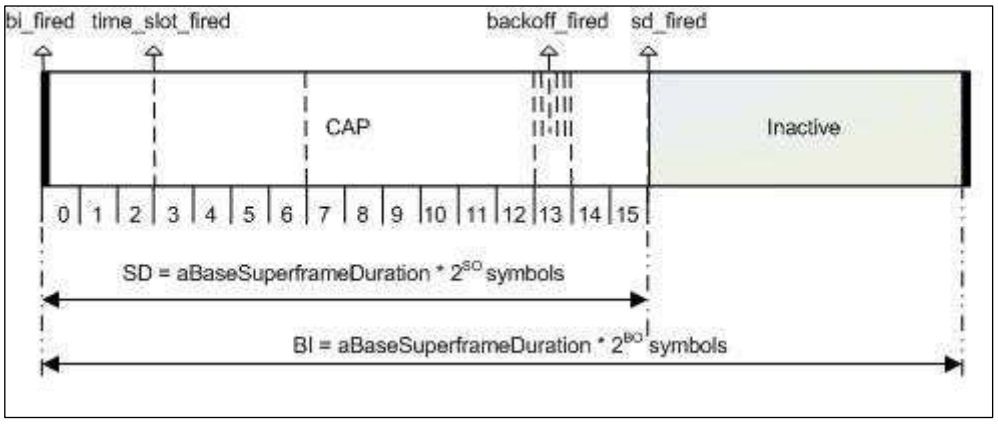


Figura 3.7 Superframe i instants d'interrupció.

mitjançant la funció ***fbucle*** que en realitza una còpia, i es passa a l'estat ***State_PD_DATA_request***, per a donar l'ordre a la capa inferior que es vol enviar un paquet. En cas de tractar-se d'un altre dispositiu comprova, si es necessari, si ha rebut un *beacon*. En cas contrari es passa a l'estat ***State_Signal_loss*** que informa a la capa superior d'aquest fet.

El següent que s'activa és ***State_TimerAsync_time_slot_fired***, igual que l'anterior confirma la interrupció i crida la funció corresponent. Aquesta és l'encarregada de processar tot el relatiu a la gestió dels GTS. Com aquesta funció és responsable de mantenir el nombre de ***slot*** actual i sap quin té assignat, és l'encarregada de passar l'execució als estats encarregats d'enviar els paquets durant els GTS. Si es tracta del coordinador l'estat de destí serà ***State_start_coordinator_gts_send***. Si no, aleshores s'activarà ***State_start_gts_send***. En el cas de que no es tingui cap GTS assignat simplement es segueix el camí per defecte, és a dir, el que no té cap condició. A la figura 3.8 es pot veure

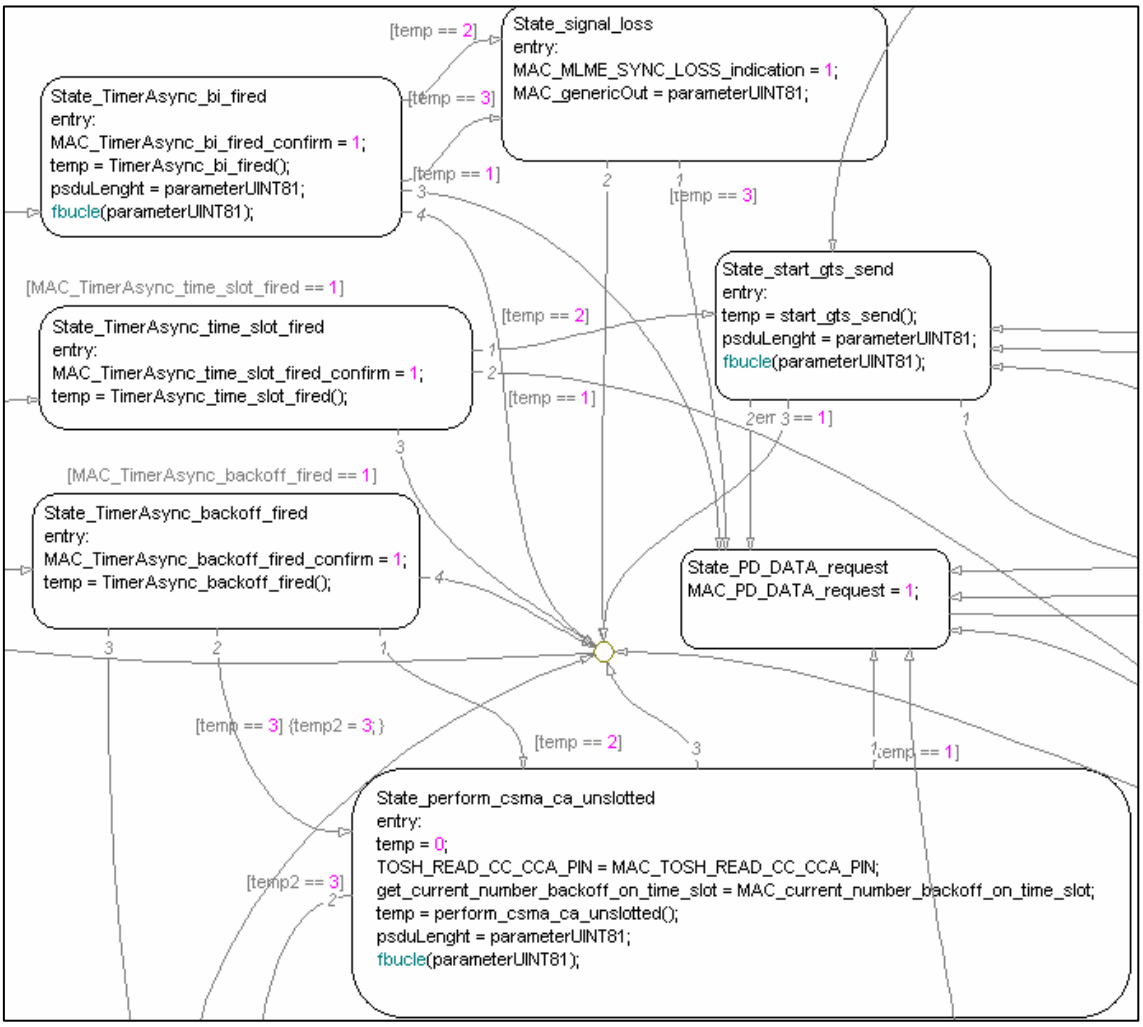


Figura 3.8 Estats corresponents a interrupcions de sincronització i part CSMA-CA

com aquesta transició porta cap a un ***junctor***, el qual és destí de totes les transicions per defecte dels estats de la imatge i més, i d'aquest es torna a l'inici.

L'última interrupció d'aquest tipus és la que ens porta a l'estat ***State_TimerAsync_backoff_fired***. Aquesta s'activa tantes vegades com pot dins d'un ***slot***. La funció que es crida es fa càrrec de la gestió de les variables que indiquen en quin punt es troba dins del ***slot***, és l'encarregada de mantenir algunes variables relacionades amb CSMA-CA i iniciar aquest procés. Tots els camins, exceptuant el per defecte, van cap a l'estat ***State_perform_CSMA_CA_unslotted*** i cap a ***State_perform_CSMA_CA_slotted.*** Aquests últims són els que realitzen el procés que acaba, si aconsegueix el canal, enviant els paquets de la cua, per això es segueix la transició que va cap a l'estat ***State_PD_DATA_request***. A la figura 3.8 s'observa que en aquest estat s'han d'actualitzar dos variables a partir dels valors d'entrada del diagrama. Això es realitza d'aquesta manera perquè són globals del codi, però en canvi no són paràmetres de la funció.

- Recepció de paquets. El procés de rebre un paquet comença amb l'activació de l'entrada ***MAC_PD_DATA_indication***, com es pot veure a la figura 3.9 aquesta té una transició que crida a la funció ***PD_DATA_indication***, la qual copia el paquet a la cua dels rebuts. El següent pas depèn de si s'està realitzant un ***scan***. Si és així, es passa a l'estat ***State_data_channel_scan_indication*** que considera el paquet un *beacon* i s'encarrega d'emmagatzemar les dades que posteriorment es donaran al nivell superior per a poder seleccionar el canal. En cas oposat, s'activa la transició que porta associada la crida a la funció ***data_indication*** que s'encarrega de distingir quin tipus de paquet és dades, comandes, *beacon* o *acknowledge*. S'ha de tenir en compte que segons l'estàndard els paquets que puguin arribar durant l'etapa de CSMA-CA han de ser ignorats.

  Als següents punts es detalla com s'actua depenent del tipus de paquet rebut. En el diagrama d'estats això està representat per les diferents transicions

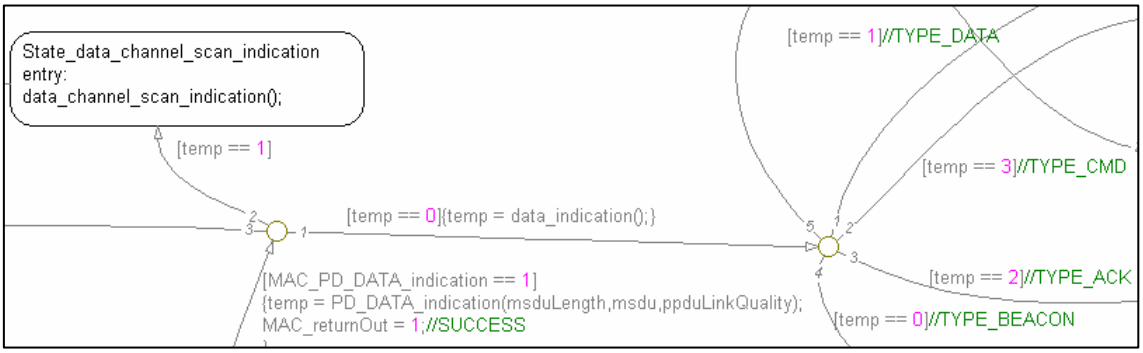


Figura 3.9 Transicions i junctors corresponents a la gestió de l'arribada d'un paquet

que surten del següent ***junctor***.

- Tractament *Beacons*. Partint del punt anterior es segueix la transició que porta a l'estat ***State_process_beacon.*** A la funció que aquest conté es processa la trama, és a dir, se n'extreu diferents tipus d'informació, com ara característiques del ***superframe***, dades del coordinador i la xarxa, o si s'ha assignat al dispositiu algun GTS.

  El següent pas depèn de si s'han donat GTS, en cas afirmatiu s'ha d'indicar a la capa superior, això en principi seria senzill, però com pot ser que se n'hagin assignat més d'un s'han de generar tantes confirmacions com GTS. El problema resideix en que només es disposa d'una sortida per a aquesta tasca. La solució resideix en el conjunt de ***transicions*** i ***junctors*** que es poden veure a la figura 3.10. El funcionament d'aquesta estructura és el següent: Prèviament, dins del codi es genera un vector amb les dades de cada confirmació. A l'estructura primer es comprova que hi ha algun element en el vector. Si és així es continua. Les transicions que surten del primer ***junctor*** determinen si la posició del vector és final, indicat amb un 2, i s'avança al següent ***junctor*** en el qual es decideix el camí depenent de la transició anterior triada, si era un final es passa a l'estat ***State_MLME_BEACON_NOTIFICATION***, que notifica el *beacon* i les seves característiques a la capa superior, i s'acaba, si no es passa al següent ***junctor*** en les transicions posteriors simplement es col·loca a la sortida l'estat del GTS

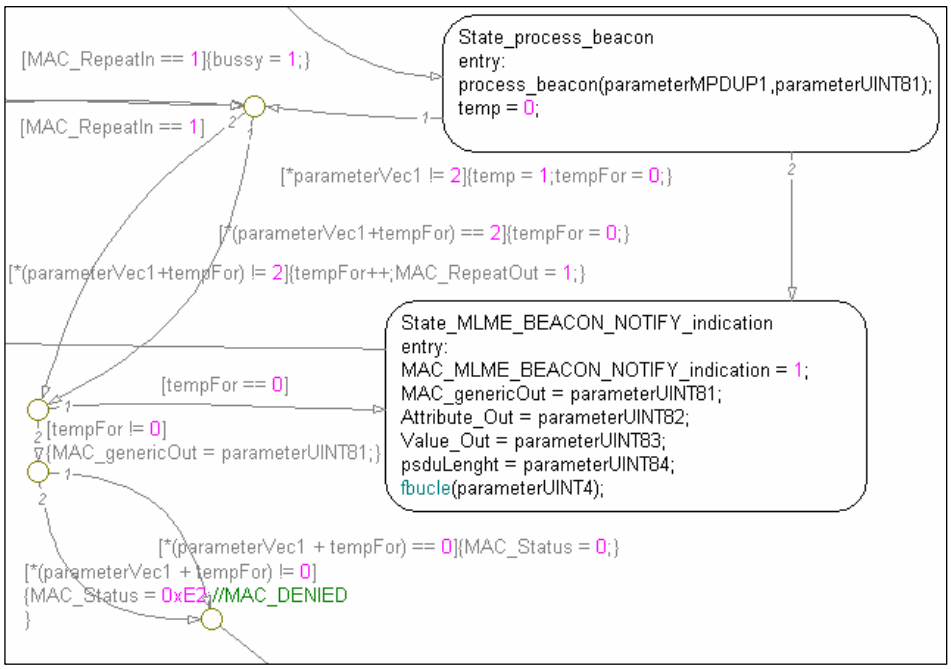


Figura 3.10 Part del processament de beacons i gestió GTS_confirm

request, és a dir, si ha estat satisfactori o no. A continuació es passa a l'estat ***State_MLME_GTS_confirm*** (veure imatge 3.13).

Finalment es torna a l'estat d'inici, però si s'observa l'acció associada a la transició amb la condició **(parameterVec1+ tempFor) != 2,* resulta que s'ha activat la sortida ***MAC_Repeat_Out***. Aquesta provoca que en la següent unitat temporal es torni, a través de la transició marcada amb ***MAC_Repeat_In***, a aquesta estructura, permetent així generar tantes confirmacions com siguin necessàries.

- Tractament de les comandes. Continuant a partir del punt de recepció de dades, aquest és un altre camí que es seguirà en cas de que el paquet rebut tingui l'estructura d'una comanda. En aquest cas, es passa a l'estat ***State_indication_cmd*** i es crida a la funció encarregada de distingir quin tipus de comanda és, depenent del resultat d'aquesta es seleccionarà la transició adequada i es farà la gestió corresponent. Les diferents opcions són:

  Association request: En aquest cas la transició porta a l'estat ***State_MLME_ASSOCIATE_indication***. En aquest es prepara l'avís a la capa superior informant de qui ha demanat ser associat i altres dades, s'activa la sortida corresponent i, si s'ha sol·licitat, s'envia un *acknowledge*, la qual cosa ja es realitza fora d'aquest.

  Association response: Si tot va correctament el dispositiu que ha enviat el ***request*** per a associar-se rebrà la comanda ***association response*** i això portarà el pas a l'estat ***State_ASSOCIATE_confirmation***. En aquest es prepara la informació que s'envia a la capa superior indicant que s'ha rebut resposta a la petició, entre aquestes dades hi ha l'estat de la petició i la nova adreça de la Personal Area Network, PAN. Si s'ha sol·licitat també s'envia el missatge de confirmació.

  Disassociation notification: Si un dispositiu vol abandonar la xarxa ho indica mitjançant aquesta comanda. L'estat que correspon a aquesta és el ***State_process_disassociation_indication_notification*** i la funció que conté és l'encarregada de preparar les dades rebudes per a poder informar la capa superior, fet que es produeix quan s'activa la sortida ***MAC_MLME_DISASSOCIATE_indication***. Com en els casos anteriors, si es requereix confirmació, s'envia.

  Data request: Aquest cas es tracta de forma diferent als anteriors ja que no té pròpiament cap acció que afecti a altres nivells. En conseqüència d'això, i de que només realitza gestió de paquets i de cues d'enviament, no té un estat

propi. Les activitats associades a aquesta petició es duen a terme en el codi i consisteixen en enviar al dispositiu que ha fet la petició els missatges que li hagi guardat mentre aquest ha estat desactivat. Però això no es pot fer directament, sinó que s'ha de seguir tot el procés d'enviament, començant pel CSMA-CA, per aquest motiu no és des d'aquí que es realitza la tasca, sinó que es posen a la cua per a ser enviats quan s'activi la interrupció corresponent al moment adequat del ***superframe***. Igual que els anteriors, també ofereix la possibilitat de respondre amb un *acknowledge*.

GTS request: Per últim aquesta comanda activa l'estat ***State_process_gts_resquest***. Des de la funció que aquest conté es preparen les dades per a enviar, a la capa superior, la informació corresponent juntament amb l'activació de la sortida corresponent per a que aquesta atengui aquestes dades.

Tots els casos anteriors donen l'opció d'enviar *acknowledges* això es realitza seguint les transicions que surten de cadascun dels anteriors estats esmentats i es dirigeixen al ***junctor*** que les agrupa per a acabar redirigint-les cap a una nova transició que genera el paquet de resposta, funció ***build_ack***, i acaba a l'estat ***State_PD_DATA_request***. Comentar que, segons l'estàndard, aquest tipus de missatges no requereixen seguir el procés que implica el CSMA-CA, per això es va directament a realitzar la petició a la capa física.

- Tractament de dades. Si el tipus de paquet que es rep indica que conté dades, aleshores l'execució evoluciona cap a l'estat ***State_indication_data***, la funció que conté aquest, simplement tracta el paquet per a extreure'n les dades i informació

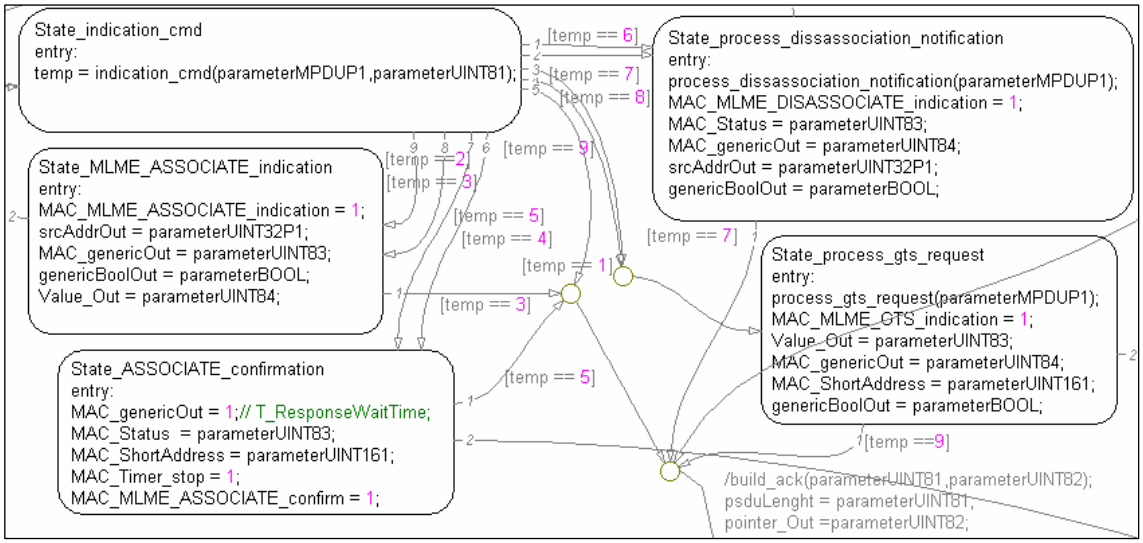


Figura 3.11 Estats relacionats amb la recepció de paquets de tipus comanda

sobre l'origen i el destí. Si no s'ha trobat cap problema s'evoluciona cap a l'estat ***State_MCPS_DATA_indication*** en el qual es posa tot el que s'ha extret del paquet en sortides de la capa i s'activa la línia ***MAC_MCPS_DATA_indication***, que avisa a la capa superior de que té disponible dades rebudes.

Com en els casos de les comandes, si es requereix enviar confirmació de recepció s'evoluciona cap al ***junctor*** comentat al final del punt anterior.

- Tractament d'*acknowledges*. En principi sembla que els "ACK" siguin fàcils de gestionar, però s'ha de tenir en compte que la recepció d'aquest tipus de paquets implica realitzar algunes operacions, com ara aturar el *timer* utilitzat per a reenviar el paquet, treure'l de la cua, iniciar altres *timers* si es tracta de la resposta d'alguna petició que ho requereix, com ara la de GTS, i algunes altres operacions.

  Segons el cas el camí a seguir es diferent, això és així fins a tal punt que s'han de preveure fins a més de 20 possibilitats. Per aquest motiu hi ha tantes transicions de sortida de l'estat que s'encarrega d'aquests paquets, ***State_indication_ack***.

  A més la complexitat augmenta degut a que es pot necessitar realitzar varies accions, per tant passar a diferents estats, quan només hi pot haver un estat

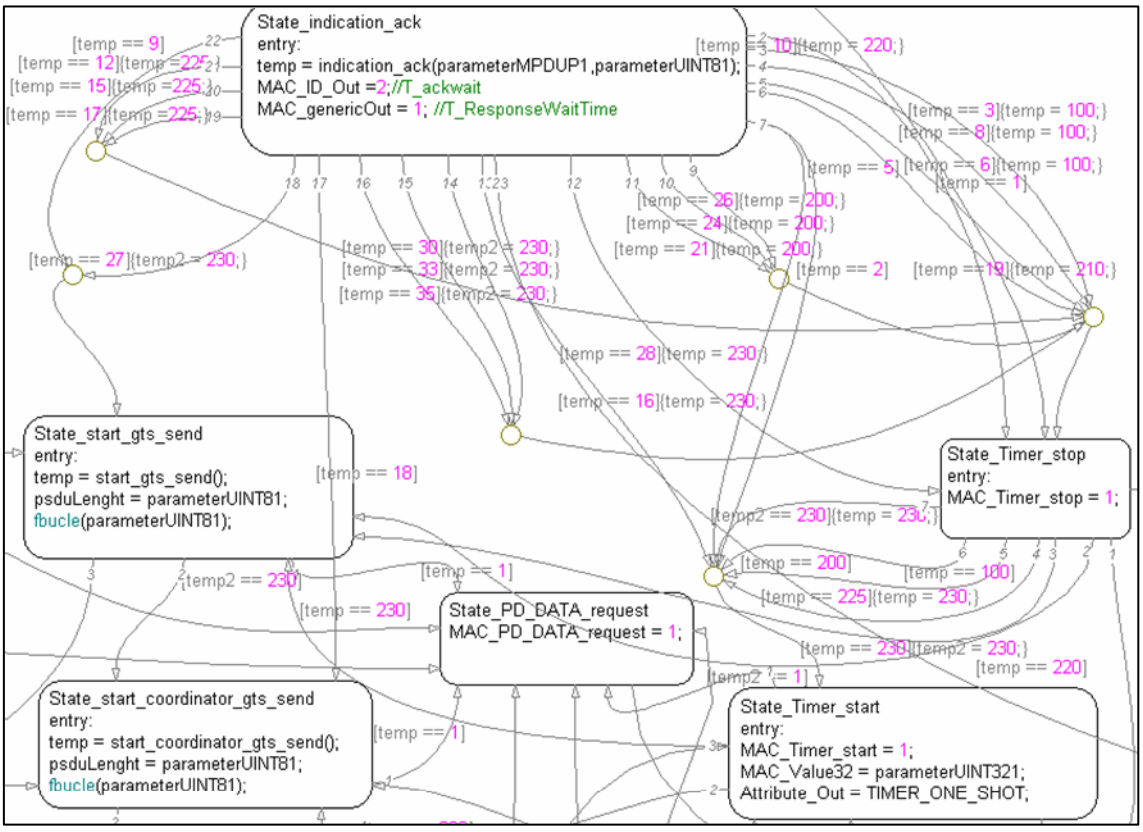


Figura 3.12 Estats i transicions relacionades amb la recepció de acknowledges

o una transició activa a cada instant. La solució a l'anterior problema passa per a crear camins que uneixin cada un d'aquests estats i que es segueixin en funció del valor retornat del codi, veure figura 3.12.

- Altres ***request***. A part de les opcions anteriors la capa superior pot realitzar altres peticions, per exemple la d'enviar dades o altres que apareixen a la figura 3.13. El funcionament d'aquestes és bastant similar en totes elles. Donades unes dades i l'activació de l'entrada oportuna es selecciona una transició que porta associada una funció, en aquesta es crea el paquet corresponent i es posa a la cua de pendents d'enviar, posteriorment seguint el procés activat per les interrupcions relacionades amb les parts del ***superframe*** es realitzarà l'enviament (veure punt interrupcions de sincronització en aquest apartat). Algunes d'aquestes peticions finalitzen activant un estat que informa a la capa superior que s'ha processat la comanda.

  Totes aquestes peticions acaben modificant variables que posteriorment, quan es rebi la resposta o com a conseqüència de la finalització d'algun *timer*, provocaran que es duguin a terme els processos que realment s'han sol·licitat, tot i que això ja no succeeix a l'entorn de la petició.

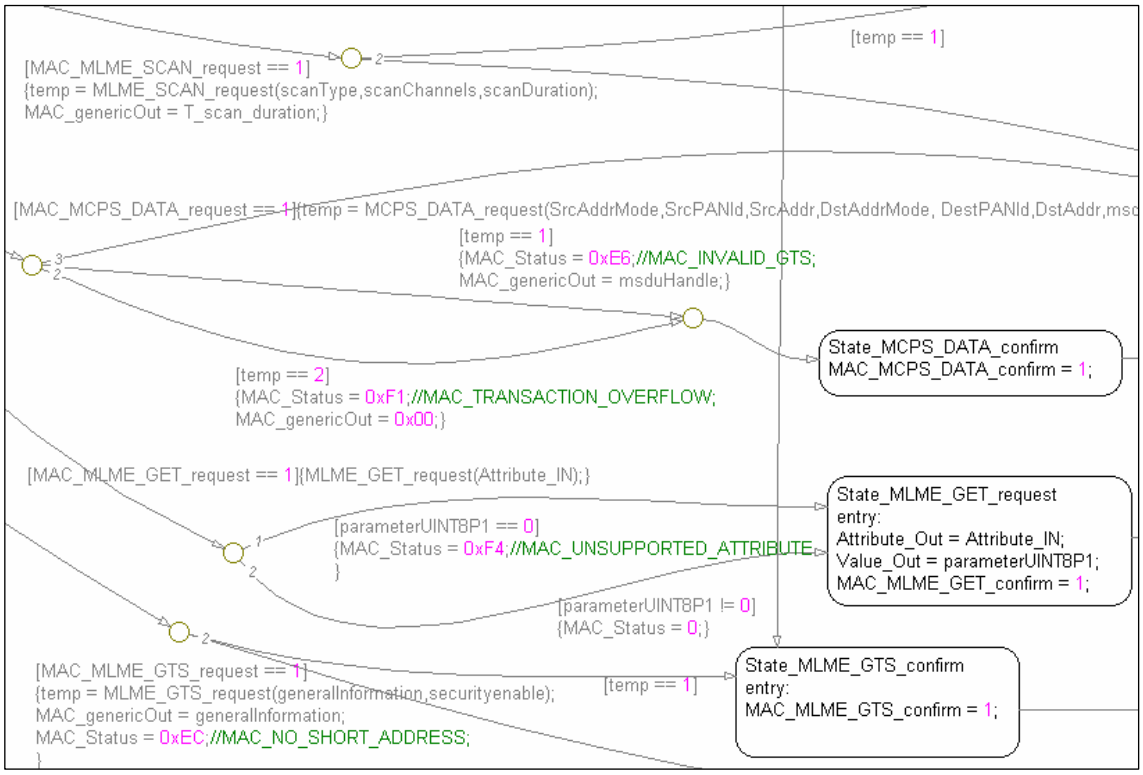


Figura 3.13 Operacions associades a les primitives Scan, Data, Get,GTS de l'estàndard

- Altres detalls de la implementació en Stateflow®. Independentment de les parts dels punts anteriors queda per comentar algunes particularitats que no formen una part en si mateixa sinó que col·laboren amb tot del diagrama.

  Prioritats d'atenció, quan s'activen diverses entrades de tipus petició en la mateixa unitat de temps es produeix un problema de selecció. Això es degut a que en Stateflow® només es pot seguir un camí a cada instant. La solució ha consistit en donar un camí per defecte format per ***junctors*** i transicions sense condició que els uneixen, a més de cada ***junctor*** surt una altra transició amb la condició corresponent a una de les entrades. Aquesta porta al camí que executa la petició. Per tant depenent de'n quin ***junctor*** es trobi la transició amb la condició, la prioritat assignada serà major o menor. Això funciona gràcies a que l'execució dels diagrames segueix el següent procés: Primer Avaluar transicions amb condició, si n'hi ha alguna de verdadera s'activa, si no n'hi ha cap amb el resultat cert es segueix la transició sense condició, en cas de que aquesta no existeixi es torna a l'estat per defecte.

  Degut a la falta de capa física i que el concepte dels temps ha variat molt entre l'execució del codi i la simulació, ha estat necessari afegir elements de sincronització entre les diferents capes i les diverses MACs que s'emulin simultàniament. Per a fer això s'ha generat una variable global que abasta tot el model de Simulink®. La funció d'aquesta és actuar com un semàfor binari permetent a un i només un agafar el testimoni en cada cas i per tant executar només el que correspongui. Per a realitzar això a les transicions que surten de l'estat inicial es comprova que aquesta variable sigui igual a '0' (com a acció associada es posa a '1') i totes les transicions que arriben a l'estat inicial tornen a deixar el valor a '0'. Aquest motiu obliga que tots els camins possibles comencin de l'estat inicial i també hi acabin. En el cas en que s'utilitza la capa física, això ja no és necessari perquè es disposa de *buffers* amb les dades que van arribant.

- Detalls de l'adaptació del codi. Com s'ha pogut observar la major part de les tasques ha de realitzar-se dins de funcions, això és degut no només a qüestions d'espai sinó també a que Stateflow® entén un conjunt molt limitat d'operacions i no contempla tipus de dades complexos. Per aquest motiu s'acaba treballant amb el llenguatge C, però això fa aparèixer un nou problema: l'intercanvi de dades. Per a dur-lo a terme es segueixen varies tècniques.

  La primera i més simple és el pas de dades a través dels paràmetres de les funcions i el valor retornat. Aquest últim es fa servir principalment per a obtenir l'indicador del camí a seguir.

La segona, tampoc molt complexa és la definició d'un conjunt de variables globals en el codi C que a més es defineixen com a externes, així s'aconsegueix que siguin visibles des del diagrama.

Última, degut a que Stateflow® no entén els punters i té problemes a l'hora de copiar vectors, copia el primer valor del vector d'origen a totes les posicions del vector destí, s'han de crear unes funcions al diagrama que s'encarreguen de copiar element per element tot l'array. Aquestes funcions són un simple bucle que a cada volta crida una funció de C, la qual aquesta li retorna el valor de l'array que està a la posició sol·licitada.

Fins aquí els detalls de la implementació de l'autòmat de la capa MAC, tot i així encara queden altres parts de suport per explicar.

3.3.4 Altres diagrames

És evident que en aquest treball la part central és el nivell MAC, però aquest no podria funcionar sense els serveis oferts principalment pels autòmats ***TimerAsync*** i ***TimerC***.

La funció del primer és gestionar les interrupcions explicades a l'apartat anterior com a interrupcions de sincronització. El diagrama d'aquest està format per un gran estat que engloba quatre estats treballant en paral·lel cadascun amb un objectiu ben definit.

Primer. Aquest conté els estats de control, és a dir, *start* i *stop*. Aquests estats activen o desactiven un altre diagrama que és l'encarregat del comptador.

Segon. La utilitat d'aquest és mantenir sempre actualitzada la sortida ***TimerAsync_current_number_backoff_on_time_slot***, per això només té un estat i una operació que es repeteix un cop per unitat de temps.

Tercer. Encarregat de desactivar les interrupcions un cop confirmades. Aquest es troba abans que l'encarregat d'activar-les perquè encara que siguin estats paral·lels els canvis de l'últim prevalen per sobre dels altres. Per a fer això es segueix un camí on a cada pas es pot triar entre una transició amb la condició de confirmació activa o una sense condició, ambdues porten a la comprovació de la següent confirmació

Quart. Si el diagrama auxiliar encarregat del comptador genera una interrupció, indicant que s'ha finalitzat el compte que s'havia requerit, es comprova si realment s'ha d'activar alguna de les línies d'interrupció, a la funció que es crida a la primera transició, i si es necessari s'activa la sortida corresponent.

Els diagrames auxiliars d'aquests s'expliquen al final d'aquest apartat.

El diagrama anomenat ***TimerC*** és responsable de les altres interrupcions, és a dir, quan donada alguna petició particular s'inicia un compte enrere que acaba amb aquestes interrupcions, si no s'ha resolt la petició quan arriba la interrupció es duen a terme una sèrie d'accions. L'estructura d'aquest diagrama és com la de l'anterior, però simplement té tres estats paral·lels, el primer encarregat de la gestió d'arrencada i parada, el segon és el que manté el compte del temps i en funció d'aquest activa les interrupcions, i el tercer s'encarrega de desactivar-les si rep la confirmació.

Cadascun dels diagrames anteriors té un altre ***chart*** auxiliar que és l'encarregat de gestionar els mòduls comptadors. Per a realitzar aquesta tasca tenen estructures similars a les anteriors, un gran estat que engloba tres estats paral·lels. Se n'explica un perquè ambdós són gairebé idèntics.

El primer estat paral·lel controla les peticions d'arrencada i parada. El *start* activa la part d'actualització dels valors i a més captura el valor actual del comptador, això serveix per a posteriorment realitzar la diferència i així poder considerar qualsevol moment com l'inicial del comptador, és a dir, no cal reiniciar el comptador.

El segon actualitza el compte, és a dir, realitza la resta entre el valor que es tenia quan s'ha començat a comptar i l'actual.

El tercer és l'encarregat d'activar la sortida d'interrupció i desactivar-la si hi ha confirmació.

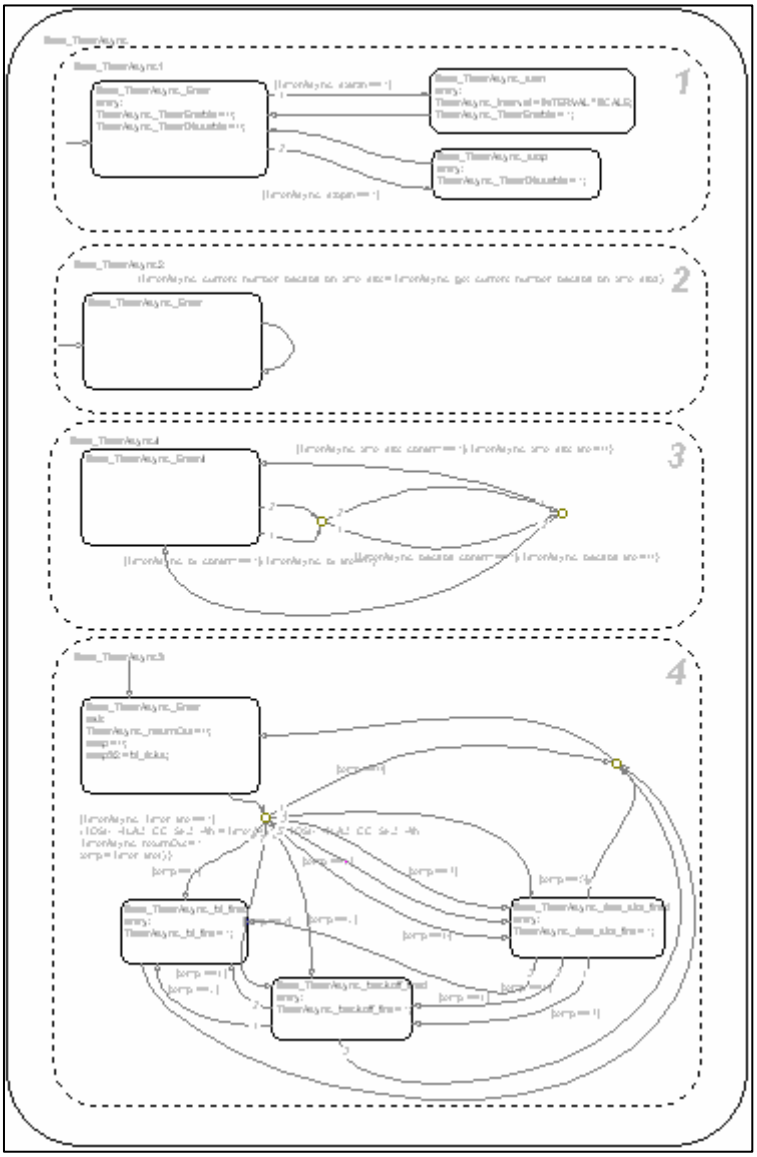


Figura 3.14 Autòmat TimerAsync

En aquest apartat i els anteriors s'anomena interrupcions a un conjunt de línies d'entrada i sortida que respecte a Simulink® no es diferencien en res a la resta, però se les considera interrupcions, perquè com les dels microcontroladors es mantenen actives mentre no s'executa la funció associada, poden arribar en qualsevol moment i es comproven cada unitat de temps.

Un cop finalitzat el desenvolupament es passa a la part de test. En aquest apartat s'ha explicat la solució final,resultat posterior a corregir els errors que s'han anat presentant, els quals es comenten amb més detall a l'apartat de problemes (veure 3.6), . És a dir, com en qualsevol procés de disseny, s'ha passat per a solucions intermitges i s'han fet modificacions fins obtenir la solució que detallem en aquest apartat.

3.4 Model de test

En aquest apartat s'explica com s'han organitzat el test i els resultats de les proves realitzades.

Per a comprovar el funcionament de la capa MAC es plantegen tres models. El primer orientat a comprovar les diferents parts de forma individual. El segon com a part de test de funcionament, és a dir, seguint el procés habitual de formar xarxa i transmetre dades. El tercer igual que l'anterior, però afegint la capa física desenvolupada per altres membres del grup, per tant es tracta d'una prova d'integració dels diferents elements.

Per al primer model es va crear un ***subsystem***, una agrupació d'elements representada per una caixa, amb les ***charts*** de la capa MAC, el ***TimerAsync*** , el ***TimerC***, els auxiliars a aquests i també altres elements de Simulink® com comptadors o generadors de nombres aleatoris. A més, es va crear un autòmat per a activar les entrades d'aquest subsistema i col·locar els valors corresponents. Un cop fetes les unions corresponents es van col·locar els *scopes* de Simulink® per a veure els resultats.

Les primeres proves van ser satisfactòries, però van fer evident que amb aquest model només es podia comprovar una part petita i poc representativa, degut a que la majoria de les parts depenen d'inicialitzacions anteriors i altres accions.

Algunes de les proves que s'han fet d'aquest tipus són: Iniciar la capa, activant l'entrada MAC_Start, s'ha pogut comprovar com s'activa l'estat i s'arrenquen alguns dels autòmats auxiliars. Activar l'entrada ***MAC_MLME_GET_request*** retorna un valor correcte i

MAC_MLME_SCAN_request genera interrupcions per ***MAC_T_Scan_duration_fired***, però la resta de proves com ara ***MAC_MLME_Start_request***, ***MAC_MLME_Associate_request*** ... no provoquen cap efecte si com a mínim no s'ha fet el procés d'activació, veure exemple a l'apartat següent.

A partir del segon model ja es va poder generar proves interessants, després del primer inici correcte, és a dir, la realització d'un ***MAC_Start*** i un ***MAC_MLME_Start_request***, ja es va començar a veure com es generaven els *beacons* i conseqüentment com es processaven correctament al destí.

Després de superar unes quantes dificultats (veure apartat 3.6), el procés d'associació va començar a funcionar, és a dir, es creava el paquet amb la comanda, es posava a la cua d'enviament i quan s'activa la interrupció corresponent s'enviava i es processava al destí.

El següent pas va ser generar un diagrama de prova que acceptés l'associació i donés l'ordre de respondre, això va ser necessari perquè és la capa superior qui s'encarrega d'això. Quan es va arribar a aquest punt ja es va poder comprovar que s'enviaven i processaven correctament els *beacons*, algunes comandes i també els *acknowledges*.

Amb la nova capa de proves i solucionats una sèrie d'inconvenients ja es generava la resposta acceptant l'associació. L'últim pas va funcionar immediatament, es va generar el paquet amb la petició de paquets pendents i quan el coordinador el va rebre va respondre el paquet de confirmació de l'associació.

Per a les següents proves es van anar afegint característiques a partir de l'anterior com ara enviament de dades o la petició de sincronització. En el moment de finalització d'aquest apartat faltava per testejar les funcionalitats de GTS i alguns camins de control en cas de pèrdua de la comunicació que no es poden provar de forma senzilla degut a que implicaria modificar el model de simulació durant la pròpia, cosa que Simulink® no permet.

La tercera fase de proves resulta igual que l'anterior, però afegint les capes físiques entre les dues MAC el resultat és igual que l'anterior degut a que aquesta capa ja estava provada i des del punt de vista del nivell MAC no ha variat res, tal com s'estableix en la comunicacions quan dues instàncies del mateix protocol intercanvien dades.

La conclusió que s'obté de les proves és satisfactòria tot i que queden parts pendents de prova que tenen complicacions. Un cop incorporada la capa física es podria haver afegit soroll al canal per a fer comprovacions extres però l'anàlisi de les dades pot ser extens i s'ha donat preferència a les que garanteixen el funcionament del projecte en condicions d'entorn favorables, ja que aquesta prova no té una durada determinada.

3.5 Exemple de funcionament

El funcionament de la capa MAC es pot seguir fàcilment gràcies a que Stateflow® ofereix un mètode de depuració molt intuïtiu, remarcant els estats i les transicions per les quals passa. L'exemple que a continuació es reprodueix s'ha extret de l'observació d'una de les últimes proves realitzades i es detalla la part més habitual de tota comunicació que utilitzi el protocol MAC de l'estàndard IEEE802.15.4.

Etapa d'iniciació, comuna per a tots els dispositius. Comença a la capa d'aplicació activant la petició de *start* relacionada amb la part hardware, és a dir, els *timers*, aquesta fa que la capa d'accés al medi posi a '1' la sortida que dona l'ordre de *start* al diagrama ***TimerAsync***. La iniciació d'aquest últim consisteix en indicar al diagrama auxiliar que controla el comptador que generi una interrupció cada temps fixat, el valor és indicat en una altra entrada d'aquest ***chart***.

Encara en la mateixa etapa, però en la unitat de temps posterior s'envia al MAC des del nivell superior la comanda per a iniciar les variables internes i altres iniciacions d'algunes parts de la capa física. Aquesta és la ***MAC_MLME_Start_request***.

A partir d'aquest punt els dispositius estan a punt per a actuar, de fet l'assignat com a coordinador de la PAN (característica indicada al pas anterior) ja envia *beacons* quan correspon.

La següent etapa pot ser la de ***scan*** o directament la d'associació, depenent de si es disposa de les dades de la xarxa a la qual es vol associar. De totes maneres, aquesta etapa només la du a terme un dels dos dispositius, el marcat com a ***end device***. El ***scan*** és una etapa que un cop testejada s'evita perquè es perd molt temps reproduint-la.

La part de ***scan*** no és molt complexa, consisteix en activar el ***TimerC***, amb la interrupció ***T_scan_duration*** que interromp una vegada per canal. En cada cas es fa una petició a la capa física per a que passi a escoltar el següent canal, un cop finalitzat, les dades que s'han rebut s'envien a la capa superior. Si arriba algun paquet durant aquest procés s'agafen algunes característiques del dispositiu que l'ha enviat i sobretot la qualitat del senyal amb que s'ha rebut.

Per altra banda, la part de l'associació és bastant més complexa i intervenen dos dispositius. És durant aquesta quan realment es produeix la primera comunicació.

Aquesta etapa comença amb la corresponent petició de la capa de xarxa o en defecte de l'aplicació. Quan el nivell MAC rep la comanda simplement es limita a generar el paquet

de tipus comanda de tipus associació i a posar-lo a la cua d'enviament. En aquests moments, la simulació no ha avançat més de 7 unitats de temps.

El següent pas és enviar el paquet, però això no succeeix fins molt després (unes 220 unitats de temps més tard) ja que s'ha d'esperar a que arribi l'instant adequat dins del ***superframe***, això s'indica per la interrupció de ***backoff***. És en aquesta on s'inicia el CSMA-CA i si no hi ha cap impediment s'acaba enviant la comanda.

Mentrestant, el dispositiu coordinador ha restat a l'espera, o ha enviat algun *beacon*, fins que arriba la petició d'associació anterior. Aquesta es processa a través de les transicions etiquetades amb les condicions *MAC_PD_DATA_indication == 1* i la *temp == 0* posterior. A les funcions lligades a les anteriors condicions es copia el missatge de la capa inferior, es posa a la cua de paquets rebuts i es diferència quin tipus de paquet és, dades, comandes, *beacons* o *acknowledges*. Si tot ha funcionat correctament l'execució evoluciona cap al tipus comanda i d'aquest a l'estat ***State_MLME_ASSOCIATE_indication*** on es prepara la informació i s'envia a la capa superior per a que decideixi si permet l'associació i amb quines dades.

Paral·lelament al procés que segueixi la capa superior per a prendre la decisió, la capa MAC segueix les transicions que la porten de l'estat anterior al ***State_PD_DATA_request***, passant per la transició que incorpora la funció ***build_ack***. Això produirà l'enviament d'un *ack* al dispositiu que ha enviat la petició.

Seguint el procés habitual, la capa de xarxa generarà un ***MAC_MLME_ASSOCIATION_response*** amb, entre altres dades, l'adreça de xarxa del nou dispositiu. Quan la MAC rep aquesta informació, genera el paquet de resposta i el posa a la cua d'enviament.

A diferència del cas anterior, aquest paquet no s'envia quan es produeix la interrupció de ***backoff*** degut a que el dispositiu receptor és un ***end device*** i teòricament podria estar aturat. És per això que no succeeix res més fins que el coordinador envia un *beacon* per l'acció de la interrupció ***MAC_TimerAsync_bi_fired*** que genera el paquet i va fins al ***State_PD_DATA_request***.

Quan l'***end device*** rep el *beacon*, seguint els passos de tot paquet fins a arribar al ***State_MLME_BEACON_NOTIFY_indication***, l'analitza. Entre la informació que porta hi ha un avís al dispositiu que té dades pendents, en conseqüència es crea un paquet de tipus comanda *data request* i es posa a la cua. Aquest s'envia seguint el procés habitual, és a dir, s'espera a la interrupció de ***backoff***, es fa el CSMA_CA i s'acaba a l'estat ***State_PD_DATA_request***.

Quan el coordinador rep el paquet, l'analitza igual que la comanda d'associació, però en lloc de passar per l'estat de notificar la comanda a la capa superior es passa directament a enviar l'*acknowledge*. En realitat, en el codi s'estan realitzant les tasques importants, que

consisteixen en accedir a la cua d'enviament indirecte de paquets, extreure'n el primer que correspongui al dispositiu que ha enviat la comanda i posar-lo a la cua d'enviament.

Mentrestant l'***end device*** haurà rebut l'*acknowledge* i l'haurà processat.

Quan es produeix la interrupció per ***backoff*** s'envia el paquet amb la confirmació de l'associació.

Finalment l'***end device*** rep aquest paquet passa per les diferents transicions fins a arribar a l'estat ***State_ASSOCIATION_confirmation*** on es preparen les sortides per a avisar la capa superior de que s'ha finalitzat el procés d'associació i l'informa del resultat.

A partir d'aquest punt ja es pot començar l'intercanvi de paquets de dades.

Enviar un paquet comença amb l'activació de ***MAC_MCPS_DATA_request*** per part de la capa superior. El funcionament és igual per a la majoria de paquets tan de dades com de comandes. Primer es crea el paquet i es posa a la cua, posteriorment s'envia amb la interrupció i el CSMA-CA. El procés de recepció és igual, però en lloc d'anar cap als estats de les comandes es va cap al de les dades i es genera una notificació al nivell superior amb les dades i altra informació. Això succeeix en els estats ***State_indication_data*** i ***State_MCPS_DATA_indication***.

Tot el procés acaba enviant un *acknowledge*, a no ser que no s'hagués demanat.

Tot i que no s'ha explicat en detall, les dades i les comandes analitzen que realment són els destinataris dels paquets un cop s'ha identificat el tipus de paquet, mentre que els *beacons* i els *acknowledges* sempre es tenen en compte, és a dir, no ho comproven.

La majoria d'operacions que es puguin realitzar segueixen el mateix procés d'intercanvi de dades.

3.6 Problemes trobats

Durant el desenvolupament de qualsevol projecte es troben problemes o inconvenients, aquest no ha estat una excepció. La diversitat de problemes trobats és important, però molts d'ells no es consideren significatius, com ara els errors de programació. A continuació es detallen tot un seguit de problemes considerats rellevants, de tipus conceptual o que suposaven un gran impediment pel seguiment del projecte, i les solucions que s'han aplicat.

- Un dels principals impediments que s'ha trobat al llarg de la implementació de l'autòmat s'ha degut a l'intercanvi de dades entre aquest i el codi escrit en llenguatge C (com ja s'ha comentat en algun punt d'aquest document).

La clau que provoca aquesta situació és que ni Simulink® ni Stateflow® accepten variables de tipus punter o bé de tipus estructures. Per a donar una idea de fins a quin punt és d'important aquest fet només cal observar que el paquet que envia i rep la capa MAC és una estructura.

Després de provar diferents alternatives sense resultat satisfactori es va optar per crear unes funcions a dins del codi que substituïen la idea del canal, per guardar les dades en una variable visible per les dues capes MAC, és a dir, quan s'enviaven les dades es guardaven en aquesta variable, quan es rebien es llegien d'aquesta.

Com que l'anterior era útil per a comprovar el funcionament, però no per al propòsit del projecte (ja que el canal no era simulat realment) es va idear la següent solució: Crear un conjunt de funcions de Stateflow® dins del propi autòmat que realitzen un bucle. Aquest itera tantes vegades com la mida de les dades que es volen aconseguir i a cada passada es crida a una funció que retorna el byte que es troba a la posició demanada com a paràmetre. D'aquesta manera es pot generar un vector a dins del Stateflow® que és una còpia del paquet o la dada creada.

- Problemes d'alineació. Aquest problema resideix en les diferències entre el compilador de nesC pensat per a microcontroladors de 8 bits i el de MATLAB® orientat a processadors de 32 bits. Això provoca diferències d'alineament a l'hora de muntar les estructures dels paquets, aquest problema genera alguns inconvenients, però són molt localitzats i es poden solucionar canviant les estructures utilitzades i adaptant-les a les noves característiques.

- Problemes de sincronització. En llenguatge C, a no ser que s'utilitzin *threads* o quelcom similar, només es té un sol punt d'execució independentment de quantes capes tingui el sistema. Mentre que en Stateflow® cada ***chart*** s'executa en paral·lel amb les altres, és a dir, a cada unitat temporal s'avalua una part de la capa MAC, una part del nivell superior, una part dels diagrames auxiliars i així amb tots els ***charts***. Això pot arribar a provocar diferents inconvenients:

  Que arribi una comanda de la capa superior quan no es pot atendre, això només amb codi no podria passar. Deixar més temps un senyal de comanda activa no és una bona solució degut a que si es finalitza el processament d'aquesta abans de que s'hagi desactivat es tornarà a repetir. També s'ha considerat la solució de crear una senyal de resposta per a cada comanda. Al final es va descartar perquè el nombre d'entrades i sortides ja es prou elevat per a tenir que afegir-ne més.

Intents d'utilitzar el mateix recurs des de diverses fonts, per exemple la capa d'accés al medi pot rebre peticions des de la capa superior i alhora des de la inferior. Sinó s'utilitza alguna estratègia per a guardar les dades d'un mentre es processen les de l'altre aquestes es perdran. Es va considerar utilitzar *buffers* entre les capes, però degut a la diversitat de les dades i que cada capa ja te els seus *buffers* es va decidir buscar una solució alternativa.

Finalment es va optar per resoldre aquests problemes amb una sola idea que els elimina tots. Aquesta consisteix en crear una variable de tipus ***Data Store Memory*** (en realitat és un bloc de Simulink®) que permet ser accedida des de qualsevol punt del model, sempre que estigui en el mateix nivell o continguda en un dels elements del mateix que la instància de la variable. Per tant, mitjançant una variable global, que té la utilitat de testimoni, es realitza la sincronització de tots els ***charts***. Quan es surt de l'estat inicial de cada diagrama que requereixi sincronització es comprova que aquesta variable estigui a '0', si es així es posa a '1' i es continua, si no s'espera. És a dir, seria l'equivalen als semàfors binaris utilitzats en sistemes operatius, però més simples ja que els diferents ***charts*** no es queden a l'espera sinó que fan *polling*.

Aquesta solució implica perdre el processament paral·lel que tenen dos dispositius diferents treballant simultàniament, però això és inevitable mentre no hi hagi capa física, degut a que es podrien perdre les dades enviades. Un cop es disposa d'aquesta capa a les hores ja es pot utilitzar una variable diferent per a les parts de cada dispositiu amb la qual cosa ja no es té l'inconvenient comentat.

- Per al problema anterior comentar que s'ha triat una solució diferent per als senyals que en llenguatge C són considerades interrupcions, o crides a funcions que es fan dins d'aquestes. Reproduint la idea de les interrupcions en un processador, s'ha pensat en un sistema que es comprovi entre totes les instruccions, en cas de produir-se s'activi un *flag*, i que aquest no es desactivi fins que no s'atengui la funció d'interrupció.

  Per a implementar això s'han pres les següents decisions: Primer, el mètode de comprovar si s'han d'activar treballa en un estat paral·lel a la resta de la gestió d'aquestes, veure apartat 3.3.4, i s'executa completament a cada unitat de temps. Segon, qualsevol senyal considerat interrupció, té el seu senyal de confirmació, per tant no es desactiven fins que no es confirmen. Últim, les

confirmacions s'activen quan s'ha començat el procés associat, és a dir, és el primer que es fa en el propi procés.

Mitjançant aquest mètode l'execució de les interrupcions queda garantida.

- El primer problema al qual encara no se li ha trobat una solució realment satisfactòria és que no es pugui copiar una capa MAC amb tots els seus mòduls auxiliars per a crear-ne una altra, és a dir, copiar un dispositiu per a crear-ne de nous.

  L'origen d'aquest problema resideix en que les instàncies de Stateflow®, tot i ser separades, tenen una base comuna i aquesta inclou tot codi extern que s'utilitzi, per exemple els diagrames de les capes superiors tenen les llibreries pròpies més totes les dels altres. Per tant, quan es creen dues entitats del mateix element estan compartint tot el codi i això implica compartir funcions i encara pitjor les variables.

  La solució, considerada provisional tot i que de moment no s'ha trobat una alternativa millor, ha estat replicar tot el codi canviant el nom a totes les variables globals i funcions que poguessin ser accedides externament. Amb aquest mètode s'aconsegueix que funcioni correctament, però no és lògic poder aplicar-lo per a crear un nombre gran de dispositius degut a que això implicaria perdre molt temps i això és totalment contrari als objectius que es persegueixen més enllà del propi projecte.

  Tot i que el treball es considera acabat i que aquest problema no l'afecta directament, es continuarà treballant per a resoldre'l en l'àmbit de la recerca, a la qual sí afecta, en que aquet projecte final de carrera s'integra.

- Ja dins dels problemes no tant importants, però que val la pena comentar s'ha de considerar un inconvenient que es va trobar al principi del desenvolupament. Aquest va resultat del fet que en Stateflow® es consumeix una unitat de temps anant d'un estat a un altre estat, per tant com a molt es poden executar la part d'***exit*** i la d'***entry*** de dos estats en una unitat. En la primera versió de l'autòmat no es va tenir en compte aquest fet i es van crear un nombre excessiu d'estats. Al adonar-nos d'aquest fet, i per una qüestió de coherència, no era lògic que tots consumissin el mateix temps, es va decidir que alguns estats no eren necessaris. Per tant, es van substituir per ***junctors*** i es van traspassar els processos que en aquests es realitzaven a les transicions anteriors o posteriors

- Altres problemes que també s'han trobat han estat que Simulink® no permet circuits tancats, molt necessaris en aquest projecte. Això s'ha solucionat afegint els ***delays*** en els senyals corresponents.

  Un altre exemple ha estat que en principi no s'havia considerat la necessitat de crear un camí de tornada des de cada estat a l'inici, cosa que ha resultat totalment necessària ja que s'han de reiniciar les sortides dels diagrames i decidir quin és el nou camí que es seguirà.

- Per a finalitzar comentar un parell de problemes relacionats amb el codi.

  Primer, durant el desenvolupament d'aquest treball ens hem adonat que la implementació de l'estàndard IEEE 802.15.4 de Open-ZB no està completa ja que hi falten algunes parts. Al no ser totalment necessàries s'han seguit altres camins existents i s'han omès les parts que mancaven.

  Segon, tal com s'ha comentat als apartats anteriors s'han tingut que fer modificacions a aquest codi, però no només per tal d'adaptar-lo al mètode de Stateflow®, sinó també per a solucionar alguns errors probablement deguts a diferències entre els compiladors del llenguatge nesC i els de C.

  Per últim fer notar la complexitat de resoldre els problemes anteriors relacionats amb el codi C, degut a que des de Stateflow® no es permet depurar més enllà de l'autòmat, per tant no es pot accedir al codi i en aquest es duen a terme parts molt importants de l'estàndard.

Fins aquí els diferents problemes trobats durant el desenvolupament del treball i amb aquests es donen per explicades les parts importants de la implementació i del resultat obtingut.

Capítol 4. Conclusions

Un cop finalitzat el treball final de carrera és el moment de tornar al principi i repassar si els resultats obtinguts han estat tal com s'esperaven en el moment de plantejar els objectius.

Repassant la feina realitza s'arriben a comptabilitzar 12 diagrames d'estats el menor dels qual conté 8 estats i 5 funcions associades, i el major té 35 estats i més de 40 funcions fent un total de més de 150 estats. El codi necessari per a fer funcionar tot aquest sistema són uns 400KB d'informació que comptat en línies de llenguatge C està per sobre de les 6000 línies, de les quals més de la meitat han estat canviades, ja sigui al passar de nesC a C, per a adaptar al funcionament de Stateflow® o també al solucionar problemes trobats.

Pel que fa a la part de verificació, s'han generat dos diagrames encarregats de testejar el funcionament des del nivell d'aplicació. Amb l'objectiu de comprovar el màxim de situacions possibles s'han arribat a crear més de 25 versions d'aquestes i s'han utilitzat en més de 100 simulacions.

El mètode per a extreure les conclusions serà recuperar els objectius un per un i avaluar si s'han acomplert. De tots els objectius que es llisten al primer capítol n'hi ha que formen part de la recerca en que aquest treball s'inclou i en principi no s'haurien de considerar, tot i així es comenten els que estan influïts per l'evolució d'aquest projecte.

- Implementació de la capa MAC de l'estàndard IEEE 802.15.4. L'objectiu no consisteix en crear un software més, sinó en aprofitar-ne algun d'existent, i reconvertir-lo per a poder treballar-hi amb ell des de l'entorn Simulink®. Dels existents no se'n coneix cap que compleixi aquests requisits.

Aquest era el principal objectiu i, vistos els resultats obtinguts, es pot donar per perfectament satisfet ja que les diferents proves han demostrat com és funcional.

- Establir la viabilitat de MATLAB® com a eina de desenvolupament de la plataforma. Aquest és un objectiu simbòlic ja que està suficientment provat que MATLAB®, amb les seves eines, permet dissenyar i crear les implementacions corresponents per a diversos dispositius.

  L'assoliment de l'anterior implica necessàriament el compliment d'aquest, degut a que ha estat MATLAB® amb les seves eines l'utilitzat per a realitzar el desenvolupament.

- Permetre accelerar el procés de desenvolupament unificant el codi simulat i el firmware dels dispositius. La majoria de simuladors existents tenen el seu propi llenguatge o estructura per a implementar els dispositius a emular. La intenció és canviar això, és a dir, oferir la possibilitat d'aprofitar al màxim la implementació de la simulació per al firmware dels dispositius. Això implica la utilització d'un llenguatge de programació per al qual la majoria de plataformes disposin de compilador. Per altra banda, un dels avantatges que es pretén aprofitar de Simulink® és la seva capacitat de generar, a partir del model, la simulació i el codi equivalent en llenguatge C. Aquesta propietat la donen les *tool-boxes* Real-Time Workshop® i Stateflow Coder ® [10].

  Aquest punt no entrava ben bé dins els objectius del treball final de carrera, ja que en cap moment es pretenia realitzar una comparació entre el temps de desenvolupament en un altre simulador i el temps utilitzat en aquest projecte. Tot i així, es pot assegurar que el codi de Open-ZB utilitzat, encara que s'hi hagin realitzat canvis, podria aprofitar-se per a programar un dispositiu real. És a dir, es pot confirmar la possibilitat de reutilitzar una gran part del codi i d'això podem extreure'n la conclusió que el temps de desenvolupament serà menor. Per tant es pot considerar aconseguit.

- Simular una xarxa amb un nombre significatiu de dispositius (en funció de la capacitat de processament). Un cop realitzat el model es pot encapsular i replicar per a formar una xarxa. Tot i que no es preveu en aquest treball, com a objectiu de la recerca s'inclou la creació tant de les capes físiques com del canal, és a dir, modelar la transmissió per l'aire i els obstacles que trobem, com ara parets, persones, etc.

Entre els problemes que s'han trobat (veure apartat 3.6) n'hi ha un, el referent a la necessitat de replicar el codi per a cada dispositiu, que impedeix que aquest objectiu es pugui complir si no es troba una solució. Per tant, queda com a treball futur d'aquest treball i com a tasca pendent de la recerca resoldre el problema.

- Al ser un projecte obert, tant en futurs objectius com en gent que hi intervindrà, les aportacions que s'hi facin s'han de poder adaptar a les possibles necessitats i mètodes de treball que es vagin requerint. Així es podrà acabar combinant des del nivell més alt, com aplicacions o estructures de xarxa, feina pròpia d'Enginyers Informàtics, fins al més baix nivell en que es troba el canal, per exemple experimentació de la transmissió a través de diferents tipus de substàncies tasca més pròpia d'un enginyer en telecomunicacions.

  Encara que no s'ha arribat a posar a prova realment fins a quin punt és realment adaptable ja s'han fet proves amb capes físiques, per tant hardware i canal. Conseqüentment es pot concloure que almenys per a l'àmbit de les comunicacions l'objectiu està assolit.

- L'entorn de simulació ha de ser el més genèric possible. Per a que usuaris de tots els àmbits es puguin involucrar en una futura evolució d'aquest projecte cal tenir present que les eines de treball han de ser molt flexibles, és a dir, han de donar els recursos necessaris per a desenvolupar qualsevol investigació o bé permetre interactuar, mitjançant interfícies, amb altres eines.

  Aquest punt forma part de la recerca, i fins que no hi comencin a intervenir gent de més àmbits no es podrà provar fins on es pot arribar. Encara que la gran diversitat de les toolbox que incorpora Simulink® fa pensar que això no serà un problema.

- Els punts restants fan referència al hardware i això ja no formava part del treball final de carrera. A més queda massa distanciat i en aquest punt extreure'n conclusions seria precipitat.

La feina que es va plantejar realitzar en aquest treball final de carrera va ser definir la viabilitat de la recerca i implementar la part essencial de protocol: la capa MAC de l'estàndard IEEE 802.15.4, crear un model a nivells de MATLAB®, i arribar a simular-lo, utilitzant Simulink®.

Com a conclusió final dir que els objectius del projecte han estat acomplerts satisfactòriament, encara que pel que fa a la viabilitat de la recerca s'hagin trobat alguns obstacles pendents de superar.

Capítol 5. Treball futur

La feina pendent d'aquest projecte és part d'una recerca que encara afegeix més tasques a realitzar. Pel que fa exclusivament al projecte es considera que s'haurien de completar els tests realitzant bancs de proves més complexos, no només amb la capa MAC, sinó també amb la física. També recordar que es va suprimir dels objectius realitzar una capa de xarxa, detall que resta pendent.

Des del punt de vista de la recerca el primer pas a realitzar és solucionar el problema relatiu a la creació de còpies del dispositiu original sense tenir que replicar els codis, amb els canvis de variables i funcions que això implica. Els passos que puguin seguir un cop resolta aquest inconvenient són molts i abasten diversos àmbits. Tot i així, es considera que la direcció a seguir és la recerca en el camp de les xarxes de sensors, primer completant una arquitectura de xarxa sencera i posteriorment començar a treballar en la descàrrega a dispositius reals.

Aquí finalitza aquest projecte, però només és el principi d'una extensa recerca.

Capítol 6. Referències

[1] Enciclopèdia Catalana S.A. GRAN ENCICLOPÈDIA CATALANA. 1986. Disponible a: http://www.enciclopedia.cat/fitxa_v2.jsp?NDCHEC=0210939

[2] The MathWorks Inc. About The Mathworks. 2008 [Online]. Disponible a : http://www.mathworks.com/company/aboutus/

[3] Enciclopèdia Catalana S.A. GRAN ENCICLOPÈDIA CATALANA. 1986. Disponible a: http://www.enciclopedia.cat/fitxa_v2.jsp?NDCHEC=0161605

[4] Enciclopèdia Catalana S.A. GRAN ENCICLOPÈDIA CATALANA. 1986. Disponible a: http://www.enciclopedia.cat/fitxa_v2.jsp?NDCHEC=0092503

[5] William Stallings COMUNICACIONES Y REDES DE COMPUTADORES (6a Edició). A.S. Prentice-Hall, 2000.

[6] Andrew Tanenbaum. REDES DE COMPUTADORAS (4a Edició). A.S. Prentice-Hall, 2003.

[7] IEEE Computer Society. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY). Specifications for Low-Rate Wireless Personal Area Networks (WPANs). 2006.

[8] Zigbee Alliance. Success Story , ZigBee Enables Control4 to Deliver Home Automation to Broad Market 2007. [Online] Disponible a: http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=10271

[9] IEEE Computer Society. IEEE 802 LAN/MAN Standards Committee Work Group 15 Task Group 4. 2008. [Online] Disponible a: http://www.ieee802.org/15/pub/TG4.html

[10] The MathWorks Inc. Mathworks Products. 2008 [Online] Disponible a: http://www.mathworks.com/products/product_listing/index.html

[11] Wolfram Research, Inc. Mathematica. 2008 [Online]

Disponible a: http://www.wolfram.com/products/mathematica/index.html

[12] **I**nstitut **N**ational de **R**echerche en **I**nformatique et en **A**utomatique (INRIA). SciLab. 2008. [Online]

Disponible a: http://www.scilab.org/

[13] **IPP** research group **HU**gging **R**eal-time and **R**eliable **A**rchitectures for computing s**Y**stems. IPP-Hurray Home page [Online]

Disponible a: http://www.hurray.isep.ipp.pt/

[14] André Cunha, Mário Alves, Anis Koubâa. IPP-Hurray. Technical Report: An IEEE 802.15.4 protocol implementation (in nesC/TinyOS): Reference Guide v1.2.

[15] David Gay, Philip Levis, David Culler, Eric Brewer. nesC 1.1 Language Reference Manual. May 2003.

Disponible a: http://nescc.sourceforge.net/papers/nesc-ref.pdf

L'estàndard IEEE802.15.4 és el més utilitzat en les xarxes de sensors. El principal objectiu d'aquest projecte és desenvolupar un model de simulació de la capa MAC utilitzant MATLAB®, les seves eines i el llenguatge de programació C.

Mitjançant aquest treball també es vol determinar la viabilitat d'una recerca d'abast més gran orientada a la creació d'una plataforma per a la simulació i la implementació d'arquitectures de xarxes principalment, però no exclusivament.

L'estructura sobre la qual es construeix aquest treball son els autòmats finits amb el suport de codi escrit en C. Tot i els problemes trobats, el resultat d'aquest projecte ha estat més que satisfactori, arribant a models complexos simulables i implementables en dispositius hardware.

El estándar IEEE802.15.4 es el más utilizado en las redes de sensores. El principal objetivo de este proyecto es desarrollar un modelo de simulación de la capa MAC utilizando MATLAB®, sus herramientas i el lenguaje de programación C.

Mediante este trabajo también se quiere determinar la viabilidad de una investigación de mayor alcance orientada a la creación de una plataforma para la simulación i la implementación de arquitecturas de redes principalmente, pero no exclusivamente.

La estructura sobre la cual se construye este trabajo son los autómatas finitos con el soporte del código escrito en C. A pesar de los problemas encontrados, el resultado de este proyecto ha sido más que satisfactorio, llegando a modelos complejos simulables e implementables en dispositivos hardware.

The standard IEEE802.15.4 is the most used in sensor networks. The main objective of this project is to develop a MAC layer model using MATLAB®, its tools and the C programming language.

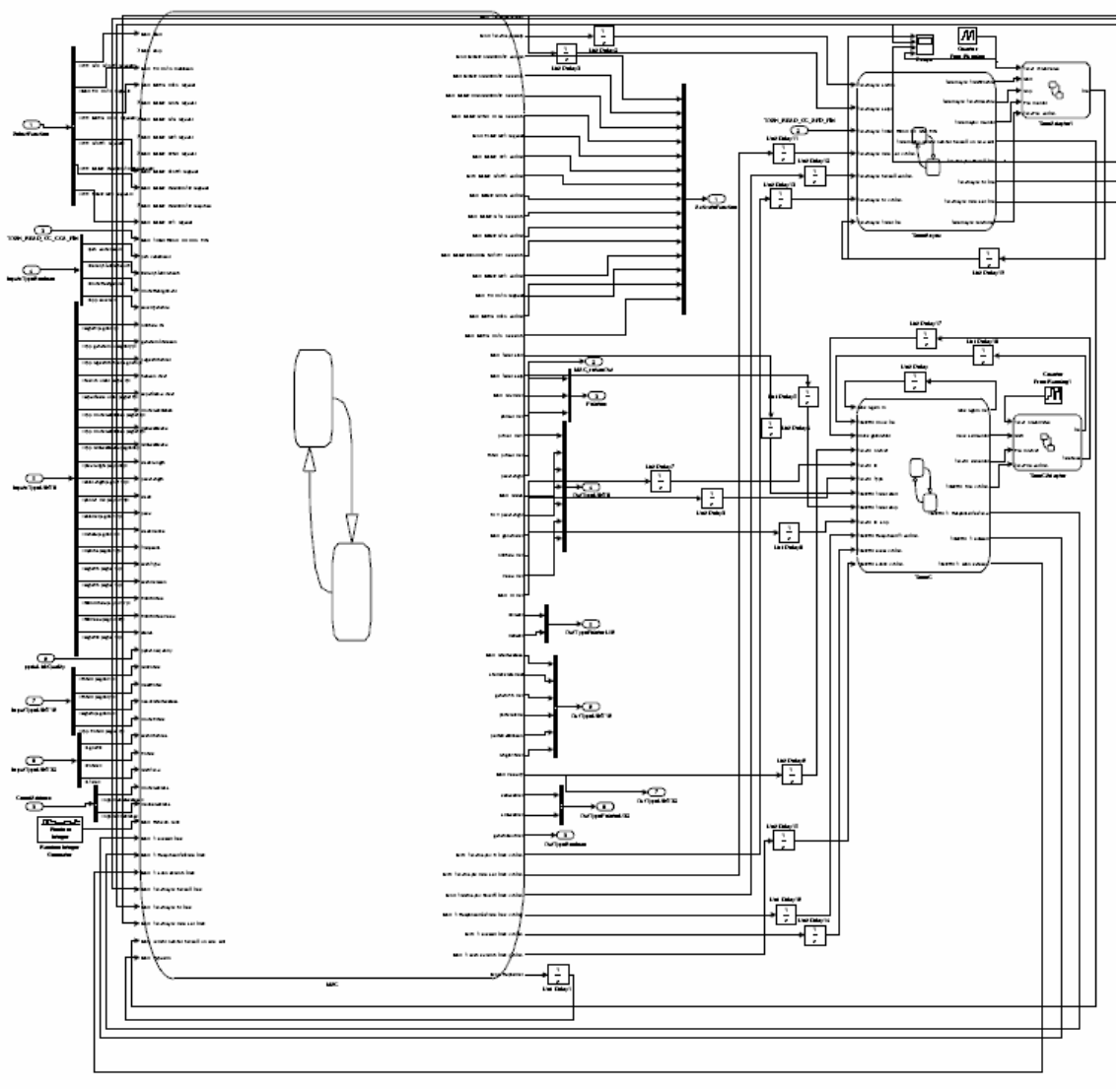
With this paper we also want to determinate the viability of a bigger investigation, which is oriented to the creation of a platform for networks simulation and implementation mainly, but not exclusively.

The structure over which this work is implemented is composed of finite-state machines and C code. Although the problems we have found, the results of this project have been more than satisfactory, reaching models which can be simulated and implemented in hardware devices.

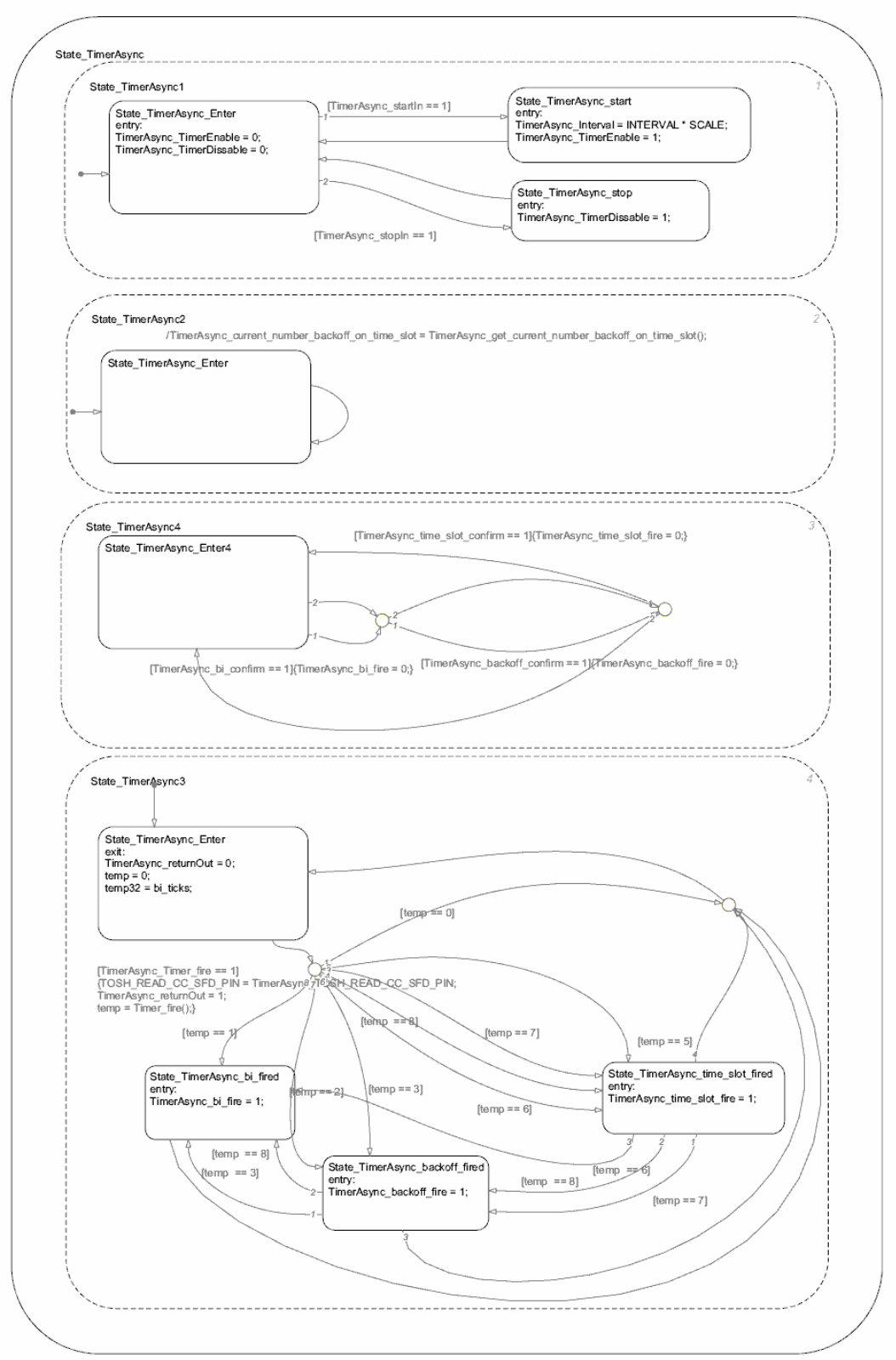
Apèndix 1. Mapes Stateflow®

A continuació es presenten els diagrames d'estats en una mida major per a poder observar els detalls. També s'inclou una visió del resultat a nivell Simulink®. Si es vol veure en una mida encara més gran en el CD adjunt s'inclouen en format PDF.

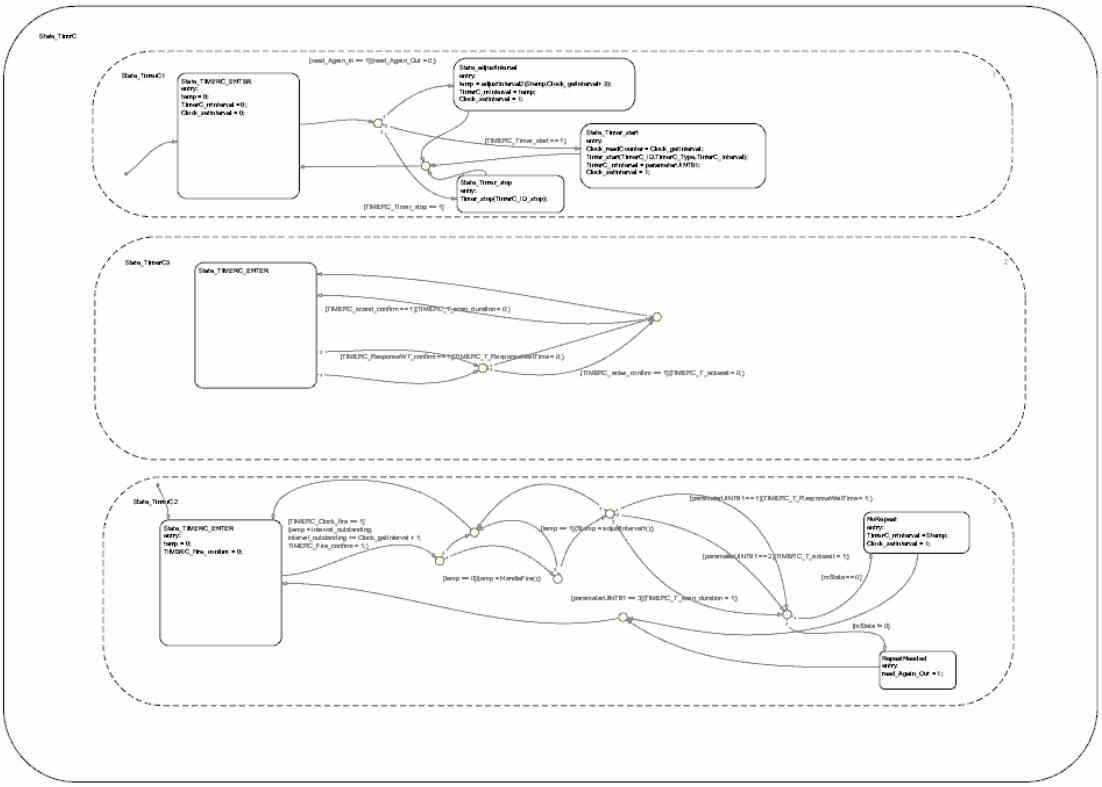
1.1 Capa MAC Simulink®



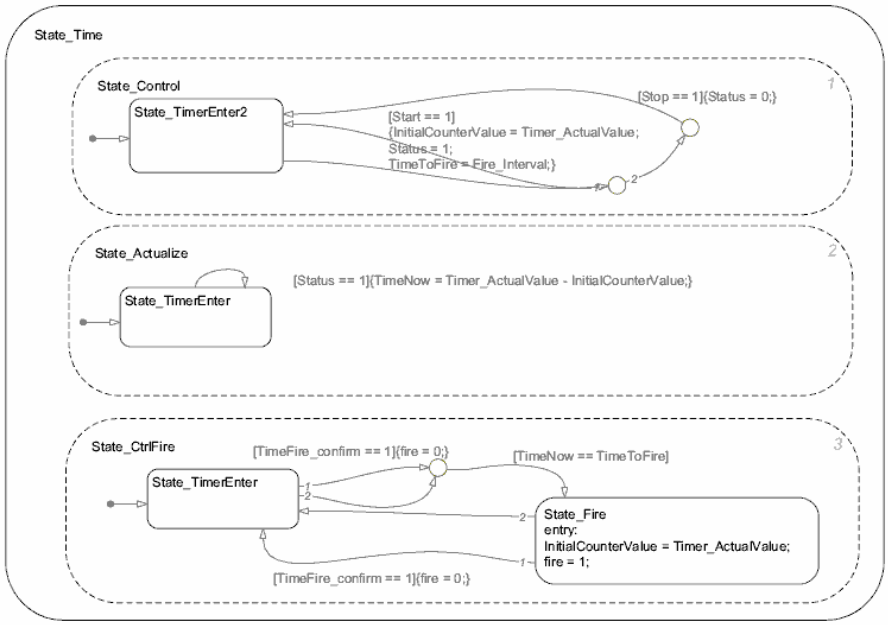
1.2 TimerAsync



1.3 TimerC

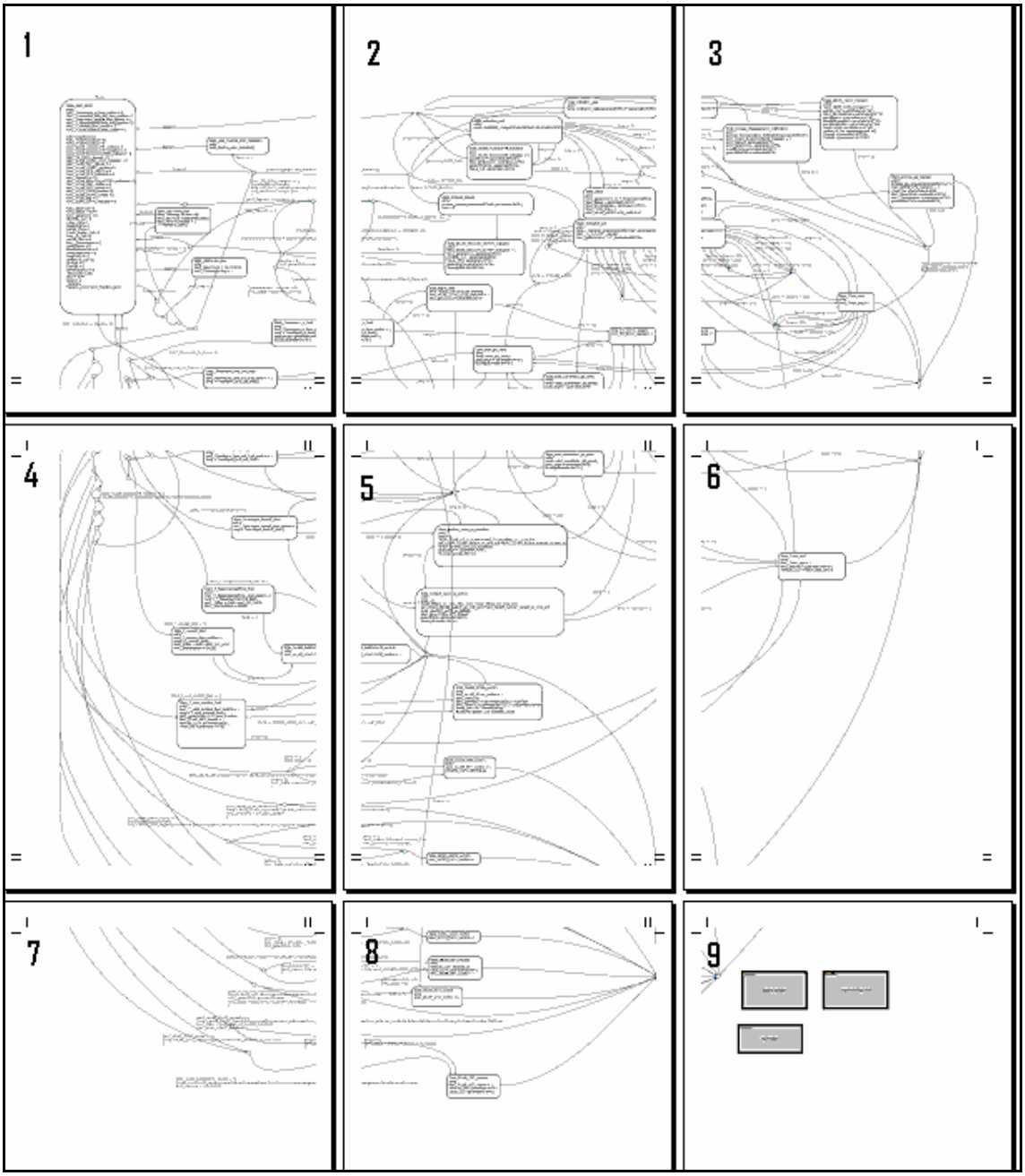


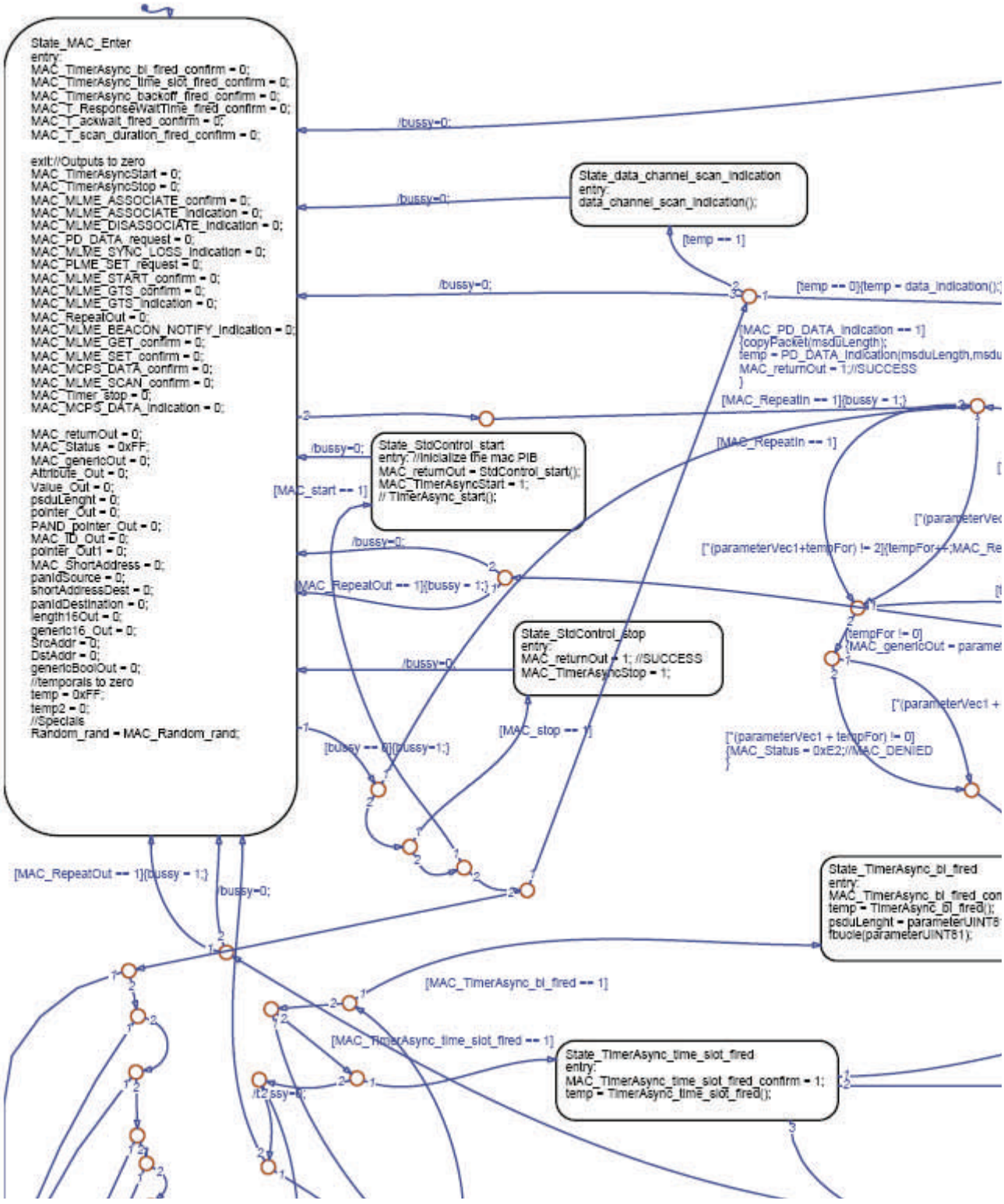
1.4 Diagrames de Suport

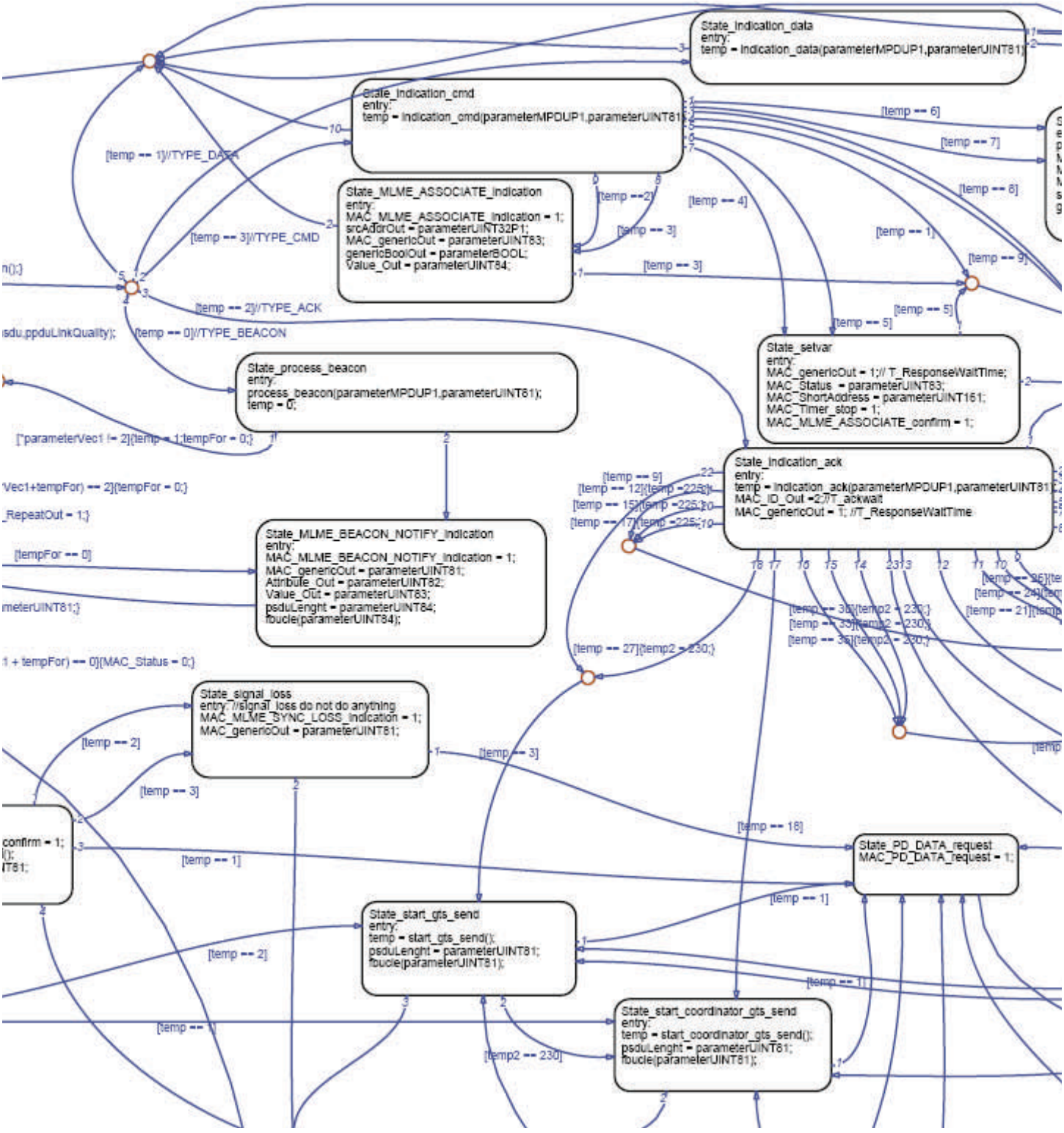


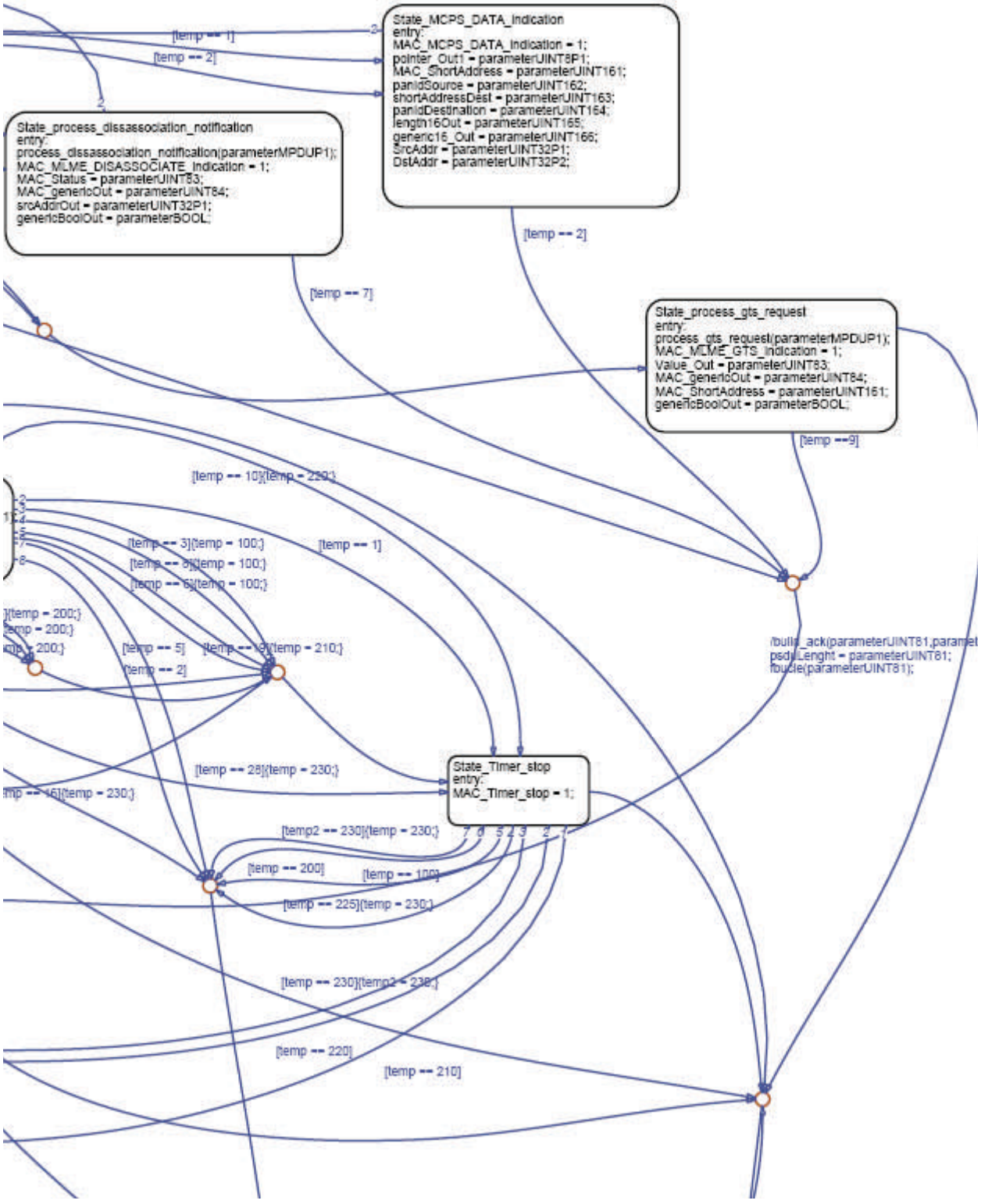
1.5 Diagrama d'estats capa MAC

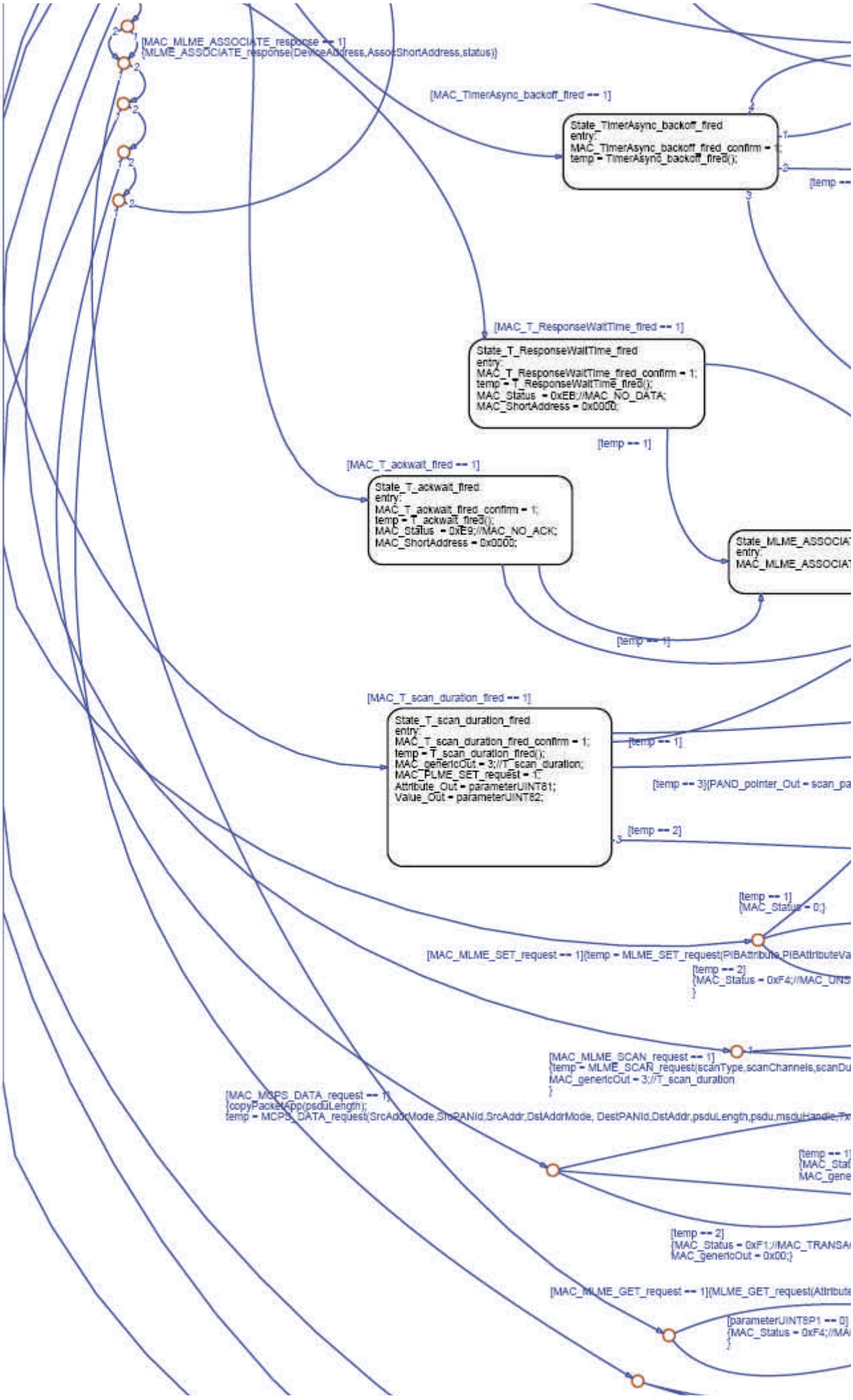
El diagrama de la capa MAC es divideix en pàgines seguint el següent esquema:

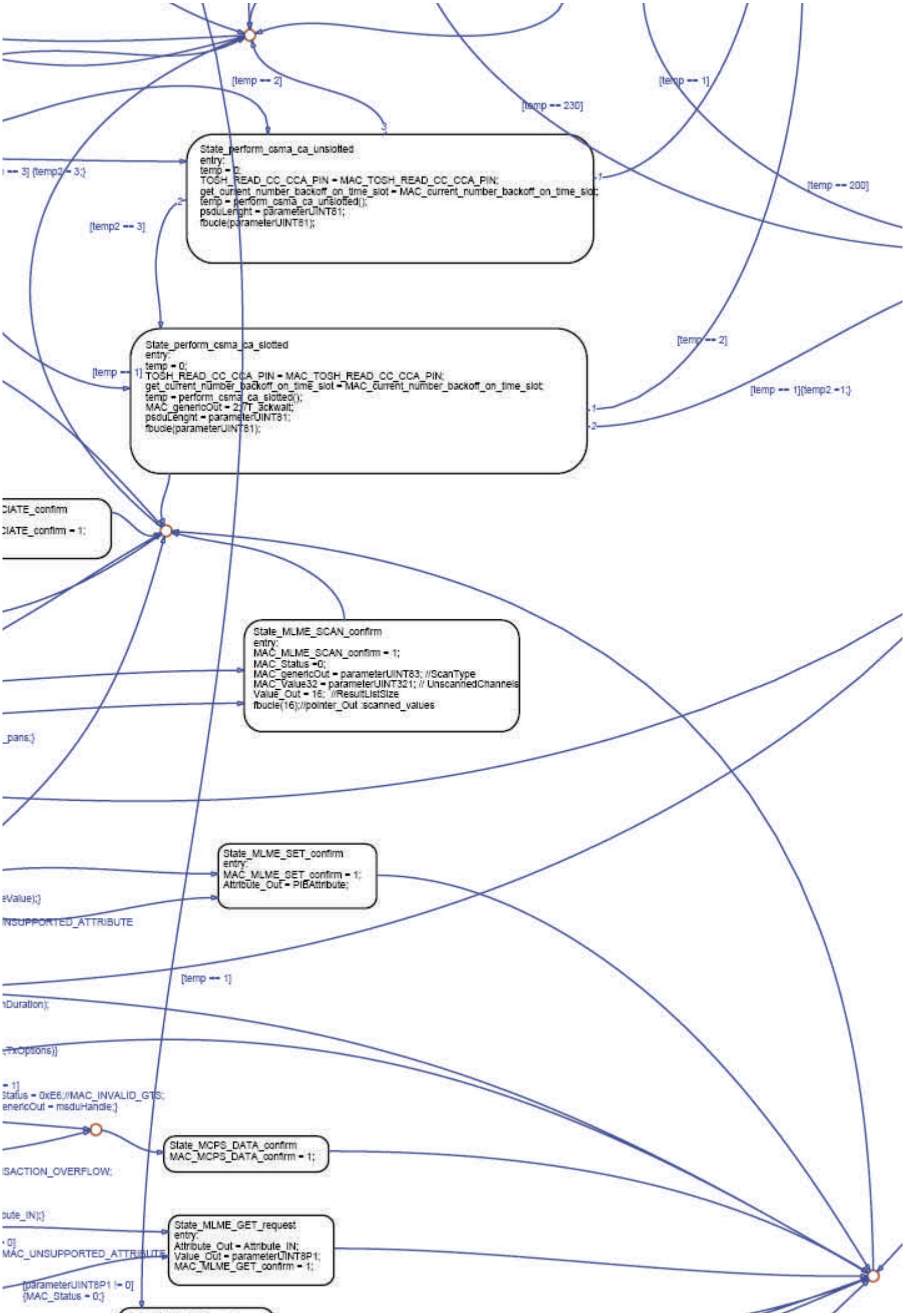


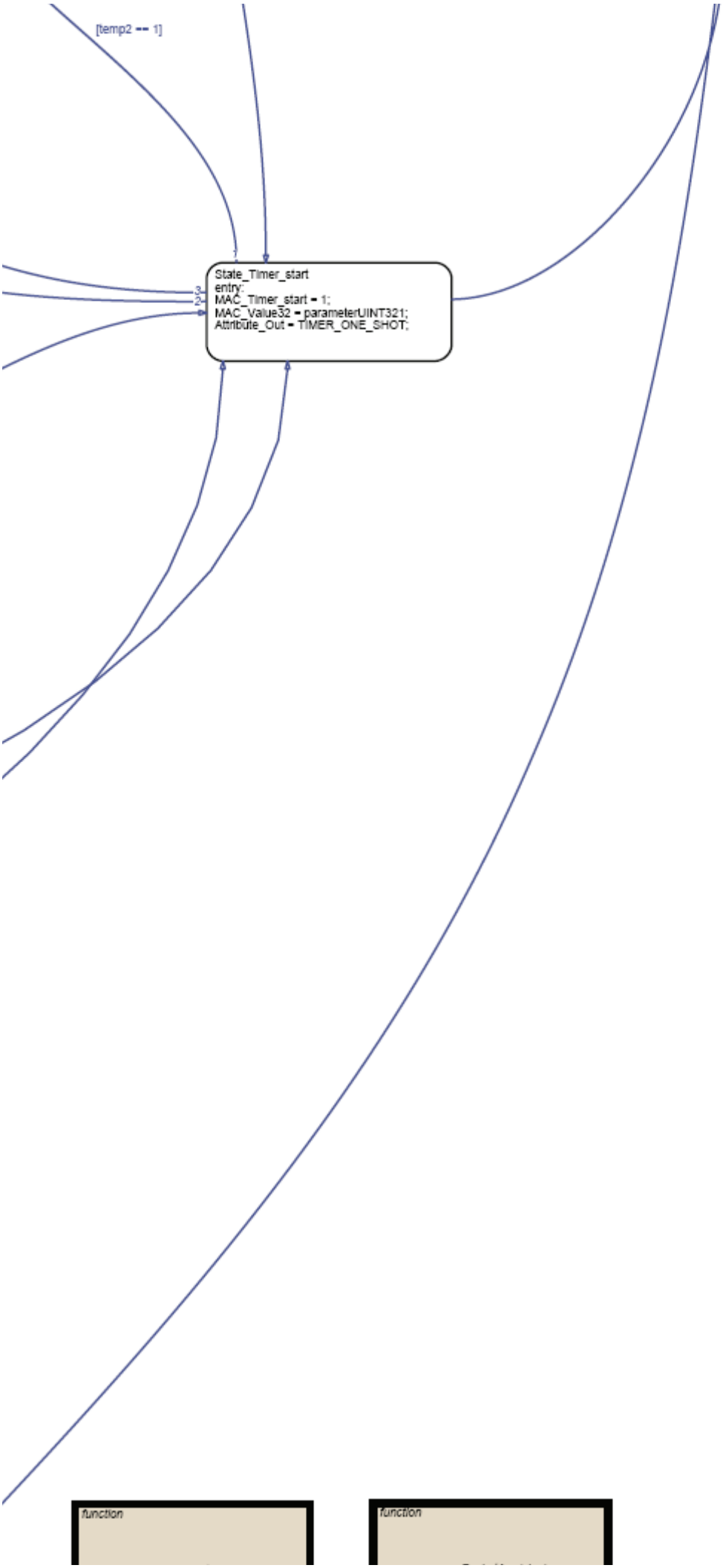

State_MAC_Enter
entry:
MAC_TimerAsync_bi_fired_confirm = 0;
MAC_TimerAsync_time_slot_fired_confirm = 0;
MAC_TimerAsync_backoff_fired_confirm = 0;
MAC_T_ResponseWaitTime_fired_confirm = 0;
MAC_T_ackwait_fired_confirm = 0;
MAC_T_scan_duration_fired_confirm = 0;
exit://Outputs to zero
MAC_TimerAsyncStart = 0;
MAC_TimerAsyncStop = 0;
MAC_MLME_ASSOCIATE_confirm = 0;
MAC_MLME_ASSOCIATE_indication = 0;
MAC_MLME_DISASSOCIATE_indication = 0;
MAC_PD_DATA_request = 0;
MAC_MLME_SYNC_LOSS_indication = 0;
MAC_PLME_SET_request = 0;
MAC_MLME_START_confirm = 0;
MAC_MLME_GTS_confirm = 0;
MAC_MLME_GTS_indication = 0;
MAC_RepeatOut = 0;
MAC_MLME_BEACON_NOTIFY_indication = 0;
MAC_MLME_GET_confirm = 0;
MAC_MLME_SET_confirm = 0;
MAC_MCPS_DATA_confirm = 0;
MAC_MLME_SCAN_confirm = 0;
MAC_Timer_stop = 0;
MAC_MCPS_DATA_indication = 0;
MAC_returnOut = 0;
MAC_Status = 0xFF;
MAC_genericOut = 0;
Attribute_Out = 0;
Value_Out = 0;
psduLenght = 0;
pointer_Out = 0;
PAND_pointer_Out = 0;
MAC_ID_Out = 0;
pointer_Out1 = 0;
MAC_ShortAddress = 0;
panIdSource = 0;
shortAddressDest = 0;
panIdDestination = 0;
length16Out = 0;
generic16_Out = 0;
SrcAddr = 0;
DstAddr = 0;
genericBoolOut = 0;
//temporals to zero
temp = 0xFF;
temp2 = 0;
//Specials
Random_rand = MAC_Random_rand;
/bussy=0;
State_data_channel_scan_indication
entry:
data_channel_scan_indication();
[temp == 1]
[temp == 0]{temp = data_indication();}
[MAC_PD_DATA_indication == 1]
{copyPacket(msduLength);
temp = PD_DATA_Indication(msduLength,msdu
MAC_returnOut = 1;//SUCCESS
}
[MAC_RepeatIn == 1]{bussy = 1;}
[MAC_RepeatIn == 1]
State_StdControl_start
entry://initialize the mac PIB
MAC_returnOut = StdControl_start();
MAC_TimerAsyncStart = 1;
// TimerAsync_start();
[MAC_start == 1]
[MAC_RepeatOut == 1]{bussy = 1;}
State_StdControl_stop
entry:
MAC_returnOut = 1; //SUCCESS
MAC_TimerAsyncStop = 1;
[MAC_stop == 1]
[bussy == 0]{bussy=1;}
[tempFor != 0]
{MAC_genericOut = parame
["(parameterVec1 + tempFor) != 0]
{MAC_Status = 0xE2;//MAC_DENIED
}
State_TimerAsync_bi_fired
entry:
MAC_TimerAsync_bi_fired_con
temp = TimerAsync_bi_fired();
psduLenght = parameterUINT8
fbuole(parameterUINT81);
[MAC_TimerAsync_bi_fired == 1]
[MAC_TimerAsync_time_slot_fired == 1]
State_TimerAsync_time_slot_fired
entry:
MAC_TimerAsync_time_slot_fired_confirm = 1;
temp = TimerAsync_time_slot_fired();

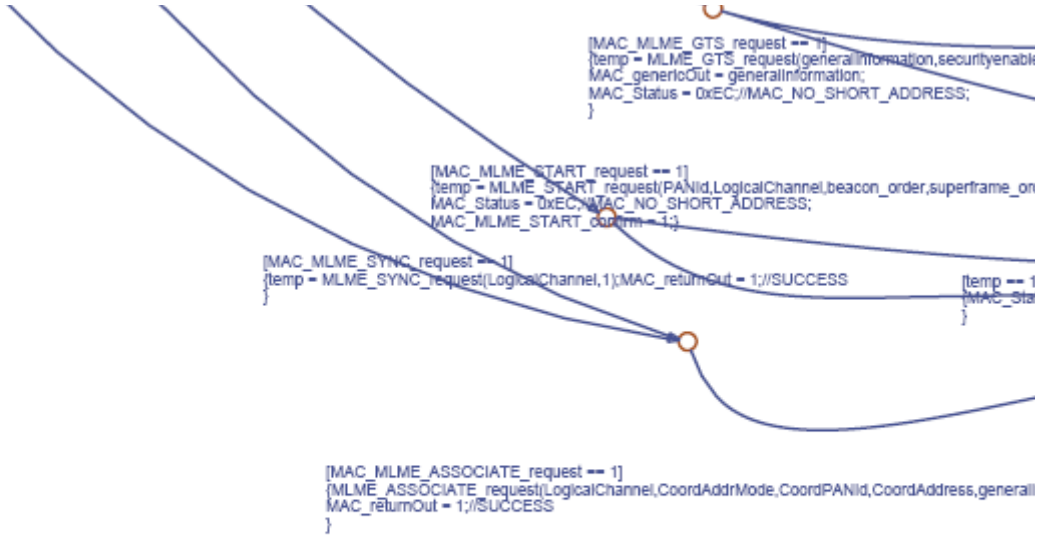

State_Indication_data
entry:
temp = Indication_data(parameterMPDUP1,parameterUINT81);
State_Indication_cmd
entry:
temp = Indication_cmd(parameterMPDUP1,parameterUINT81);
State_MLME_ASSOCIATE_Indication
entry:
MAC_MLME_ASSOCIATE_Indication = 1;
srcAddrOut = parameterUINT32P1;
MAC_genericOut = parameterUINT83;
genericBoolOut = parameterBOOL;
Value_Out = parameterUINT84;
State_process_beacon
entry:
process_beacon(parameterMPDUP1,parameterUINT81);
temp = 0;
State_setvar
entry:
MAC_genericOut = 1;// T_ResponseWaitTime;
MAC_Status = parameterUINT83;
MAC_ShortAddress = parameterUINT161;
MAC_Timer_stop = 1;
MAC_MLME_ASSOCIATE_confirm = 1;
State_Indication_ack
entry:
temp = Indication_ack(parameterMPDUP1,parameterUINT81);
MAC_ID_Out =27//T_ackwait
MAC_genericOut = 1; //T_ResponseWaitTime
State_MLME_BEACON_NOTIFY_Indication
entry:
MAC_MLME_BEACON_NOTIFY_Indication = 1;
MAC_genericOut = parameterUINT81;
Attribute_Out = parameterUINT82;
Value_Out = parameterUINT83;
psduLenght = parameterUINT84;
boucle(parameterUINT84);
State_signal_loss
entry: //signal_loss do not do anything
MAC_MLME_SYNC_LOSS_Indication = 1;
MAC_genericOut = parameterUINT81;
State_PD_DATA_request
MAC_PD_DATA_request = 1;
State_start_gts_send
entry:
temp = start_gts_send();
psduLenght = parameterUINT81;
boucle(parameterUINT81);
State_start_coordinator_gts_send
entry:
temp = start_coordinator_gts_send();
psduLenght = parameterUINT81;
boucle(parameterUINT81);
[temp == 1]//TYPE_DATA
[temp == 2]//TYPE_ACK
[temp == 3]//TYPE_CMD
[temp == 0]//TYPE_BEACON
[temp == 6]
[temp == 7]
[temp == 8]
[temp == 9]
[temp == 4]
[temp == 5]
[temp == 27][temp2 = 230;]
[temp == 16]
[temp2 == 230]

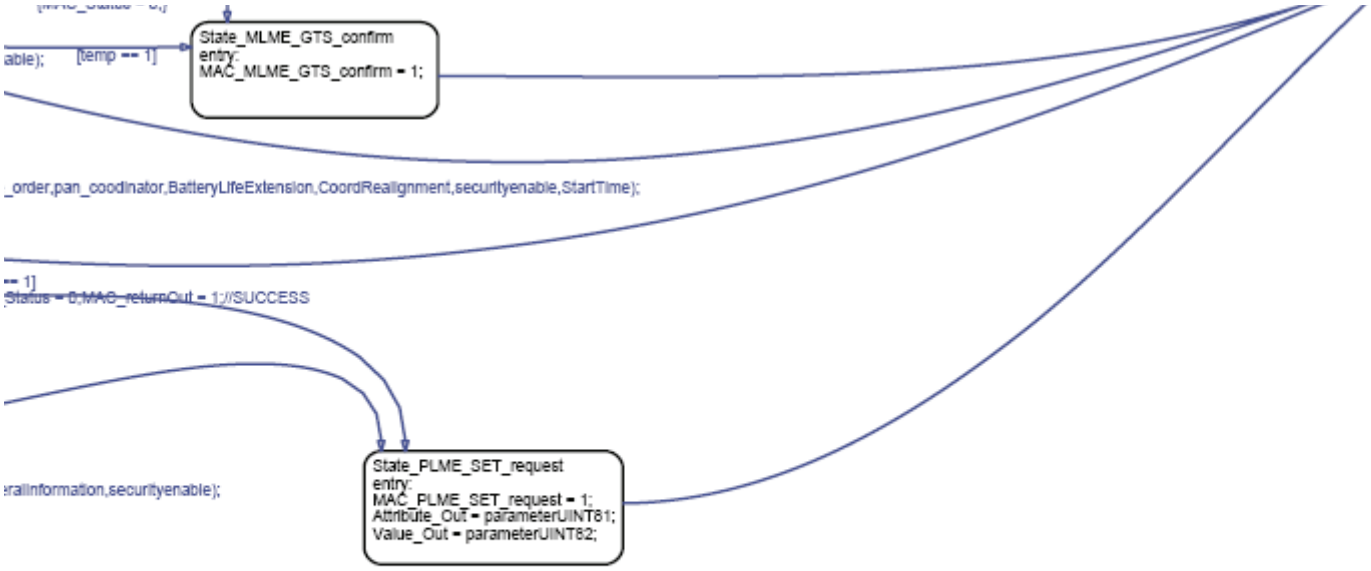

State_MCPS_DATA_Indication
entry:
MAC_MCPS_DATA_Indication = 1;
pointer_Out1 = parameterUINT8P1;
MAC_ShortAddress = parameterUINT161;
panIdSource = parameterUINT162;
shortAddressDest = parameterUINT163;
panIdDestination = parameterUINT164;
length16Out = parameterUINT165;
generic16_Out = parameterUINT166;
SrcAddr = parameterUINT32P1;
DstAddr = parameterUINT32P2;
State_process_dissassociation_notification
entry:
process_dissassociation_notification(parameterMPDUP1);
MAC_MLME_DISASSOCIATE_Indication = 1;
MAC_Status = parameterUINT83;
MAC_genericOut = parameterUINT84;
srcAddrOut = parameterUINT32P1;
genericBoolOut = parameterBOOL;
State_process_gts_request
entry:
process_gts_request(parameterMPDUP1);
MAC_MLME_GTS_Indication = 1;
Value_Out = parameterUINT83;
MAC_genericOut = parameterUINT84;
MAC_ShortAddress = parameterUINT161;
genericBoolOut = parameterBOOL;
State_Timer_stop
entry:
MAC_Timer_stop = 1;
[temp == 1]
[temp == 2]
[temp == 7]
[temp == 9]
[temp == 10][temp = 220;]
[temp == 3][temp = 100;]
[temp == 8][temp = 100;]
[temp == 6][temp = 100;]
[temp == 5]
[temp == 2]
[temp == 28][temp = 230;]
[temp2 == 230][temp = 230;]
[temp == 200]
[temp == 100]
[temp == 225][temp = 230;]
[temp == 230][temp2 = 238;]
[temp == 220]
[temp == 210]


[MAC_MLME_ASSOCIATE_response == 1]
{MLME_ASSOCIATE_response(DeviceAddress,AssocShortAddress,status);}
[MAC_TimerAsync_backoff_fired == 1]
State_TimerAsync_backoff_fired
entry:
MAC_TimerAsync_backoff_fired_confirm = 1;
temp = TimerAsync_backoff_fired();
[temp ==
[MAC_T_ResponseWaitTime_fired == 1]
State_T_ResponseWaitTime_fired
entry:
MAC_T_ResponseWaitTime_fired_confirm = 1;
temp = T_ResponseWaitTime_fired();
MAC_Status = 0xEB;//MAC_NO_DATA;
MAC_ShortAddress = 0x0000;
[temp == 1]
[MAC_T_ackwait_fired == 1]
State_T_ackwait_fired
entry:
MAC_T_ackwait_fired_confirm = 1;
temp = T_ackwait_fired();
MAC_Status = 0xE9;//MAC_NO_ACK;
MAC_ShortAddress = 0x0000;
State_MLME_ASSOCIA
entry:
MAC_MLME_ASSOCIA
[temp == 1]
[MAC_T_scan_duration_fired == 1]
State_T_scan_duration_fired
entry:
MAC_T_scan_duration_fired_confirm = 1;
temp = T_scan_duration_fired();
MAC_genericOut = 3;//T_scan_duration;
MAC_PLME_SET_request = 1;
Attribute_Out = parameterUINT81;
Value_Out = parameterUINT82;
[temp == 1]
[temp == 3]{PAND_pointer_Out = scan_pa
[temp == 2]
[temp == 1]
{MAC_Status = 0;}
[MAC_MLME_SET_request == 1]{temp = MLME_SET_request(PIBAttribute,PIBAttributeVa
[temp == 2]
{MAC_Status = 0xF4;//MAC_UNS
}
[MAC_MLME_SCAN_request == 1]
{temp = MLME_SCAN_request(scanType,scanChannels,scanDu
MAC_genericOut = 3;//T_scan_duration
}
[MAC_MCPS_DATA_request == 1]
{copyPacketApp(psduLength);
temp = MCPS_DATA_request(SrcAddrMode,SrcPANId,SrcAddr,DstAddrMode, DestPANId,DstAddr,psduLength,psdu,msduHandle,Tx
[temp == 1]
{MAC_Stat
MAC_gene
[temp == 2]
{MAC_Status = 0xF1;//MAC_TRANSA
MAC_genericOut = 0x00;}
[MAC_MLME_GET_request == 1]{MLME_GET_request(Attribute
[parameterUINT8P1 == 0]
{MAC_Status = 0xF4;//MA
}


[temp == 2]
[temp == 230]
[temp == 1]
State_perform_csma_ca_unslotted
entry:
temp = 0;
TOSH_READ_CC_CCA_PIN = MAC_TOSH_READ_CC_CCA_PIN;
get_current_number_backoff_on_time_slot = MAC_current_number_backoff_on_time_slot;
temp = perform_csma_ca_unslotted();
psduLenght = parameterUINT81;
fbucle(parameterUINT81);
[temp == 200]
[temp2 == 3]
State_perform_csma_ca_slotted
entry:
temp = 0;
TOSH_READ_CC_CCA_PIN = MAC_TOSH_READ_CC_CCA_PIN;
get_current_number_backoff_on_time_slot = MAC_current_number_backoff_on_time_slot;
temp = perform_csma_ca_slotted();
MAC_genericOut = 2;//T_ackwait;
psduLenght = parameterUINT81;
fbucle(parameterUINT81);
[temp == 2]
[temp == 1]{temp2 =1;}
State_MLME_SCAN_confirm
entry:
MAC_MLME_SCAN_confirm = 1;
MAC_Status =0;
MAC_genericOut = parameterUINT83; //ScanType
MAC_Value32 = parameterUINT321; // UnscannedChannels
Value_Out = 16; //ResultListSize
fbucle(16);//pointer_Out :scanned_values
State_MLME_SET_confirm
entry:
MAC_MLME_SET_confirm = 1;
Attribute_Out = PIBAttribute;
[temp == 1]
State_MCPS_DATA_confirm
MAC_MCPS_DATA_confirm = 1;
State_MLME_GET_request
entry:
Attribute_Out = Attribute_IN;
Value_Out = parameterUINT8P1;
MAC_MLME_GET_confirm = 1;
[parameterUINT8P1 != 0]
{MAC_Status = 0;}

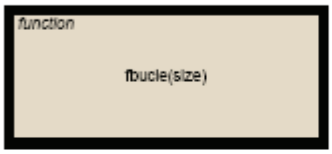

[temp2 == 1]
State_Timer_start
entry:
MAC_Timer_start = 1;
MAC_Value32 = parameterUINT321;
Attribute_Out = TIMER_ONE_SHOT;
function
function


[MAC_MLME_GTS_request == 1]
{temp = MLME_GTS_request(generalInformation,securityenable
MAC_generlcOut = generalInformation;
MAC_Status = 0xEC;//MAC_NO_SHORT_ADDRESS;
}
[MAC_MLME_START_request == 1]
{temp = MLME_START_request(PANId,LogicalChannel,beacon_order,superframe_or
MAC_Status = 0xEC;//MAC_NO_SHORT_ADDRESS;
MAC_MLME_START_confirm = 1;}
[MAC_MLME_SYNC_request == 1]
{temp = MLME_SYNC_request(LogicalChannel,1);MAC_returnOut = 1;//SUCCESS
}
[temp == 1
{MAC_Sta
}
[MAC_MLME_ASSOCIATE_request == 1]
{MLME_ASSOCIATE_request(LogicalChannel,CoordAddrMode,CoordPANId,CoordAddress,generalI
MAC_returnOut = 1;//SUCCESS
}


able);
[temp == 1]
State_MLME_GTS_confirm
entry:
MAC_MLME_GTS_confirm = 1;
_order,pan_coodinator,BatteryLifeExtension,CoordRealignment,securityenable,StartTime);
== 1]
Status = 0;MAC_returnOut = 1;//SUCCESS
rallinformation,securityenable);
State_PLME_SET_request
entry:
MAC_PLME_SET_request = 1;
Attribute_Out = parameterUINT81;
Value_Out = parameterUINT82;


copyPacket(size)


copyPacketApp(size)


function
fbucle(size)